FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Never Ending Language Metalearning: model management for CMU's ReadTheWeb project

Tiago Vieira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Carlos Soares

Second Supervisor: Estevam Hruschka Jr.

July 20, 2015

Never Ending Language Metalearning: model management for CMU's ReadTheWeb project

Tiago Vieira

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Professor Gabriel David
External Examiner: Professor Nuno Escudeiro
Supervisor: Professor Carlos Soares

July 20, 2015

Abstract

The main goal of CMU's ReadTheWeb project is to build a new kind of machine learning system that continuously reads the web, 24 hours per day, 7 days per week. This system is called the Never-Ending Language Learner (NELL) [Carlson et al., 2010a]. While this goal is not necessarily unheard-of, NELL stands out as being capable of improving the way it learns over time, that is to say, it learns to read the web better than it did the day before. To succeed in such an arduous quest, NELL combines several subsystem components that implement complementary knowledge extraction methods and for the same task, is able to use different extraction methods. However, the performance of the components that use such methods, that is the quality of the extracted knowledge for different topics, will however change over time. Futhermore, the evaluation of the produced knowledge is not immediate, and can, sometimes, take days or even months to be performed. This means that the approach of monitoring the system to identify models that are not performing as well as expected is far from optimal, as any corrective measure will be made long after it was required. We follow a predictive approach, in which we try to how good or bad the quality of the produced knowledge is, at any given time. A preliminary approach to use metalearning to address this issue was proposed by dos Santos [2014]. This approach sought to relate the innate (meta)features of the data produced and the performance of the models generated with it. In this project, we extend this work. In order to do so, the range of components studied was expanded and new metafeatures, that needed to to fit every component, developed. An extensive emperical study, comprised of an exploratory data analysis and two experiments with different sets of metafeatures, was then performed. The created metalearning system proposed in this work, is able to, for different components, predict the quality of the data produced for different topics with satisfactory accuracy. This result indicates that this is a viable approach to improve the learning ability of NELL.

Resumo

O principal objetivo do projeto ReadTheWeb da CMU é desenvolver um novo tipo de sistema de aprendizagem que lê a web continuamente, 24 horas por dia, 7 dias por semana. Este sistema é chamado de "Never-Ending Language Learner" (NELL) [Carlson et al., 2010a]. Embora este objetivo não seja necessariamente novo, a NELL destaca-se como sendo capaz de melhorar a forma como aprende ao longo do tempo, o que equivale a dizer que lê a web melhor hoje do que leu no dia anterior. Para ser bem sucedido nesta árdua tarefa, a NELL combina vários componentes de subsistema que implementam métodos de extração de conhecimento complementares e para uma mesma tarefa, é capaz de usar diferentes métodos de extração. A performance, no entanto, dos componentes que usam tais métodos, isto é a qualidade do conhecimento extraído para diferentes tópicos, irá variar ao longo do tempo. Por outro lado, a avaliação do conhecimento produzido não é imediata e pode, por vezes, levar dias ou até meses a ser efectuada. Isto implica que técnicas de monitorização do sistema para identificar os modelos que não se comportam como esperado está longe de ser óptima já que qualquer medida correctiva será efectuada muito depois de ser necessária. Neste trabalho usamos técnicas preditivas em que tentamos, a qualquer momento, estimar quão bom ou mau é o conhecimento produzido. Uma abordagem preliminar usando meta-aprendizagem para combater este problema foi já proposta por dos Santos [2014]. Nessa abordagem procurou-se relacionar as (meta)características dos dados e a performance dos modelos gerados por ela. Este projeto propõe-se a estender esse trabalho. Para isso, o número de componentes estudados foi aumentado e novas metacaracterísticas, que conseguissem representar os diferentes componentes, desenvolvidas. De seguida, foi então efectuado um trabalho empírico que passou por uma primeira análise dos dados seguida de duas experiências com diferentes conjuntos de metacaracteristicas. O sistema de meta-aprendizagem proposto neste trabalho é capaz, para diferentes componentes, prever a qualidade da informação proposta para diferentes tópicos com precisão satisfatória. De uma forma geral, os nosso resultados mostram que as técnicas usadas neste trabalho podem ser aplicadas no NELL para melhorar sua a capacidade de aprendizagem.

Acknowledgements

First of all, I would like to thank my supervisors Carlos Soares and Estevam Hruschka for their support and dedication. I would also like to thank Bryan Kisiel for his advice and support. His help and knowledge were crucial to the advance of the project. A special thank you is also in order for everyone at SAPO Labs for their warm welcome and helpfullness over these last few months.

To my parents, my brother and my friends I leave a truly heartfelt thank you. I would not be the person I am today without them.

Lastly, but definitely not least, I would like to thank my partner Joana for her support and endless patience. Without her, nothing like this would have ever been possible.

Thank you,

Tiago Vieira

Contents

CONTENTS

List of Figures

List of Tables

Abbreviations

CMU	Carnegie Mellon University
NELL	Never-Ending Language Learner
ALICE	Architecture-Neutral Distribution Format
POS	Part of Speech
CPL	Coupled Pattern Learner
CSEAL	Coupled Set Expander for Any Language
CML	Coupled Morphological Learner
PRA	Path Ranking Algorithm
OntExt	Ontology Extension
NEIL	Never Ending Image Learner
k-NN	k-Nearest Neighbors
SVM	Support Vector Machine
CART	Classification And Regression Tree
PPR	Projection Pursuit Regression

Chapter 1

Introduction

In the last decade the increase of available online information and the advances of machine learning and natural language brought forth new opportunities in the field of information extraction from text. Due to the amount and diversity of the information, extracting knowledge from such a source requires an ongoing and complex learning process. This process if often referred as lifelong learning or never-ending learning, and has been explored by NELL [Carlson et al., 2010a, Mitchell et al., 2015], a computer system developed at Carnegie Mellon University (CMU). In a different line of research, a new paradigm of machine learning, Metalearning, has shown real potential with dealing with vast amounts of information [Brazdil et al., 2008]. Metalearning is capable of empowering learning systems by exploiting previous experiences and adapting the learning process to the available data at hand. In this project, the two areas are combined. In this chapter, we will start by summarizing the problems of lifelong learning systems and how one could use metalearning to improve them and give an overview of the structure of the report.

1.1 Motivation

The paradigm of lifelong learning was first introduced to shorten the distance between machine and human learning abilities [Silver et al., 2013]. However, the developed systems tend to increase in complexity over time and require multiple learning tasks to work simultaneously. This increase in complexity and the everlasting amount of knowledge to be learned are serious obstacles that must be overcome.

In particular, the complexity of the system makes it hard to ensure that all of the models it uses are performing in a satisfactory way. At the time, NELL still lacks the ability to correctly assess the performance of its components over different topics in real time [Mitchell et al., 2015]. That is to say that, the information produced at one moment might take several days (maybe even months) to be assessed as true or not.

It is thus necessary to invest in a solution that is capable of monitoring the different components of these systems and make intelligent decisions without jeopardizing valuable computing time.

1.2 Project

An approach that has been used to monitor and update learning models is metalearning. Metalearning consists of using a machine learning approach to create models that relate the characteristics of a learning problem with the performance of a learning algorithm [Brazdil et al., 2008]. By leveraging on the huge amount of information produced by NELL, a metalearning solution will be devised so as to evaluate the quality of information produced by the system. With this new found power, NELL will be able to self-regulate and better manage its different components.

A preliminary approach to using metalearning in the context of a state-of-the-art lifelong learning system, NELL, was proposed by dos Santos [2014]. In this project, we seek to extend this work. Our solution, however, covers more than one component (four to be exact) and presents new metafeatures that fit the data produced by such components in a generalized fashion. Furthermore, an extensive empirical study was performed that encompassed a preliminary data analysis followed by two separate experiments with different sets of metafeatures.

1.3 Structure of the Report

The next four chapters of this report will explore these above mentioned topics and the work that needs to be performed. In chapter 2 we review the field of knowledge extraction and its influence in machine learning techniques. In this same chapter, we will then introduce NELL and its architecture as well as concepts of metalearning and their connection with knowledge extraction. In chapter 3 we will explore the basis of our solution and in chapter 4 we will discuss its implementation and results. Finally, in chapter 5 we will conclude on the work done and present future work opportunities to be explored.

Chapter 2

State-of-the-Art

In this chapter, we will begin with an overview of Knowledge Extraction and its algorithms, with reference to its applications in machine learning tasks. We will then review NELL and its individual components and introduce Metalearning in the context of algorithm selection. Lastly, a previous attempt at mixing the aforementioned fields will be shortly described.

2.1 Knowledge Extraction

Throughout the last decades researchers have tried to coin several terms for knowledge extraction. Such terms include the popularized "data-mining" as well as less common ones like "pattern analysis" and "data dredging". In Jiawei Han et all the authors describe the process of knowledge extraction as

> "...the process of discovering interesting patterns and knowledge from large amounts of data. The data sources can include databases, data warehouses, the Web, other information repositories, or data that are streamed into the system dynamically " [Han et al., 2011]

2.1.1 Knowledge Extraction tasks

While there is a multitude of different algorithms used in knowledge extraction we can find four major groups of tasks shared between them. Those are classification, regression, clustering and association. As a whole, these tasks can be described as either descriptive or predictive. Descriptive tasks deal with characterizing the properties of data and predictive tasks use data to make predictions about unlabeled data [Flach, 2012].

Classification is probably the most common task among the four described above. In a classification task one tries to construct a classifier (also referred as a model) which distinguishes data classes. This classifier uses training data (previously known instances) so that future instances

where the class label is unknown can be identified. In the simplest case where there are only two classes (True or False, for example) it is called it binary classification. For any given instance, this classifier must be able to estimate the correct target class as best as possible. While there are many types of classifiers, linear, Bayesian and distance-based classifiers should be acknowledged as being the most used [Flach, 2012]. These can take the form of neural networks [Cheng and Titterington, 1994], support vector machines [Hearst et al., 1998] and k-nearest-neighbors [Friedman et al., 1975] to name a few.

Decision trees can also be seen classifiers, although there are various ways of representing such knowledge (such as classification rules or mathematical equations). In decision trees each node represents a test on a feature value and the branch represents the outcome of such test. Finally the leaves of the structure represent the class prediction. As in classification, in regression one tries to to create a model capable of making a prediction based on previously given data. However, while classification deals with discrete labels, in regression labels take the form of a numerical value [Han et al., 2011].

While classification and regression tasks are concerned with analyzing data labels, clustering deals with grouping data without considering previously defined labels. In clustering the instances of data are bundled together in groups that maximize intraclass similarity and minimize the interclass similarity. That is to say that the objects in each cluster are more alike to each other than the others in any other computed cluster. It is then up to the analyst to label each cluster according to its properties and extract useful knowledge from the result. Clustering methods can take the form of partitioning, hierarchical, density-based and grid-based. In the first method, the challenge is to create partitions which are then iteratively improved over an objective function. An example of such method is k-means [Jain et al., 1999].

In hierarchical clustering one aims, as indicated by the name, at decomposing a given set of data objects in a hierarchical fashion. This can lead to improvements in computation cost at the cost of being unable to correct erroneous decisions. Density-base methods introduce the notion of arbitrarily shaped clusters that are not present in other partitioning methods (whose clusters are only spherical-shaped). Lastly, grid based methods try to form a grid-like structure over the object space which can then be used in other clustering methods. This reduces the computational cost involved, since the methods are now dependent only on the number of cells and not the number of data objects [Han et al., 2011].

In an association task one tries to identify items that frequently occur together [Flach, 2012]. One simple example of association rules lies in the prediction of what items go together in a market basket. In this case the algorithm tries to identify which product will be bought next considering a set of already bought products. In classification this could be compared to a case where at any given time one has to predict, for every object, the likelihood of that product being bought considering any combination of different (previously) selected products. While association rules can be seen in many different areas such as Web site navigation analysis [Cooley et al., 1999] and medical diagnosis [Delgado et al., 2001] this task as been widely associated with product recommendation [Kotsiantis and Kanellopoulos, 2006].

2.1.2 Machine learning applications

While discussing knowledge extraction, it is important to cover its applications in machine learning. Most learning tasks in this area can be classified as unsupervised, supervised, semisupervised or active learning. In unsupervised learning our data is not class labeled and one strives to find potential labels that match our problem. This is usually done by clustering. In supervised learning we provide labeled examples, hence the term supervised. This kind of learning is often associated with classification tasks where from an initial training set (labeled data) one tries to create a model which can then infer the correct labeling of future instances. Semi-supervised learning tries to make use of both labeled and unlabeled data, using the second to refine the model created from the first. One example of this kind of learning is bootstrap learning. This is commonly used in relation extraction on the web where we have large amounts of unlabeled data and only a couple of known instances known as seeds [Chen et al., 2006]. This model, however, is prone to errors which will be further discussed in section 2.2. Lastly, in active learning, the user is expected to label a number of given instances in order to optimize the quality of the model. In section 2.2 we will discuss a type of active learning aptly named "Conversing Learning".

2.1.3 Evaluation

Finally, it is important to discuss how the performance of the models created by the above mentioned algorithms is measured and how to deal with problems like overfitting (where a model performs very well over the given training data but is incapable of generalizing for unknown data). It is thus good practice to separate the available data over a training set with which one creates a model and a testing set on which one evaluates the model performance. There are many ways one could divide the training and testing sets [Han et al., 2011] but the following are the most common:

Hold-out

Training and testing sets are simply separated according to a fixed ratio (usually two thirds of the available data are used for training) and the performance is scored over the remaining data.

Cross-validation

With cross-validation one repeatedly follows the same procedure of Hold-out for different sets and averages the resulting performances. In k Fold cross-validation one divides the original set in k folds and uses one fold at a time for testing (and every other fold for training).

Leave-one-out

This technique, useful for smaller datasets, focuses on leaving only one instance of the dataset for testing and using every other instance for training. This is done for every instance and is considerably more expensive computationally wise.

These techniques allow us to have a more detailed perspective about possible overfitting problems but still required standardized measures of performance. Most of these measures relate in

one way or another to the difference between the expected and predicted values [Willmott, 1982]. In regression these are, but not limited to: Mean Bias Error (MBE) which is simply the average of the difference between predicted and target value for every instance; Mean Square Error (MSE), the squared MBE; Root Mean Square Error (RMSE), the root squared MBE and Mean Absolute Error (MAE) which uses the absolute value of the difference between predicted and target values. While the last two measures have been widely used, there is still some debate on whether it is best to use one over the other [Willmott and Matsuura, 2005].

It is also important to notice that these measures, on their own, are either hard to interpret or bring no usable information. While a value of zero is ideal, in different contexts, some errors could be described as very high or very low. It is thus common practice to compare these measures against measures of very simple models (like the average or median of the training data) that represent the most human like model one could create without recourse to machine learning algorithms. These simple models are commonly known as base models, and a useful measure of performance could be:

$$RRMSE = \frac{RMSE_{model}}{RMSE_{base}} \tag{2.1}$$

Where $RMSE_{model}$ is the RMSE of the model created and $RMSE_{base}$ the RSME of the base model we just described. Following this formula, a RRMSE value closer to one means that our model is not much better than the base model and a result closer to zero means a good prediction compared to that made by human like prediction. If, however, the value of RRMSE is higher than 1, then our model predicts worse than the base model and should be discarded. In [Gangemi, 2013] the authors review the available tools for knowledge extraction on the semantic web. This area will be further explored in section 2.2 where we will also discuss knowledge extraction when dealing with lifelong learning [Banko and Etzioni, 2007]. Automatic knowledge extraction has also been proposed by [Alani et al., 2003] in the creation of Artequakt, a system capable of extracting information about artists from multiple documents based on a predefined ontology.

2.2 NELL

In this section we will cover NELL, a semi-supervised computer system that learns to read the web. NELL, also known as Never-Ending Language Learner, is part of a research project aptly named "Read the Web" at Carnegie Mellon University. The main goal of this project is to develop a never-ending machine learning system that runs 24 hours per day, 7 days per week and is able to, not only extract information from unstructured web pages, but also to learn how to perform this task better each day [Carlson et al., 2010a]. While the concept of never ending machine learning is not new [Silver et al., 2013, Small and Medsker, 2014], NELL faces practically no competition from other systems since it started running, in 2010. In [Banko and Etzioni, 2007] the authors describe a lifelong learning agent called ALICE that is able to build a collection of concepts from large volumes of web data which can then be used to focus its search for additional information.

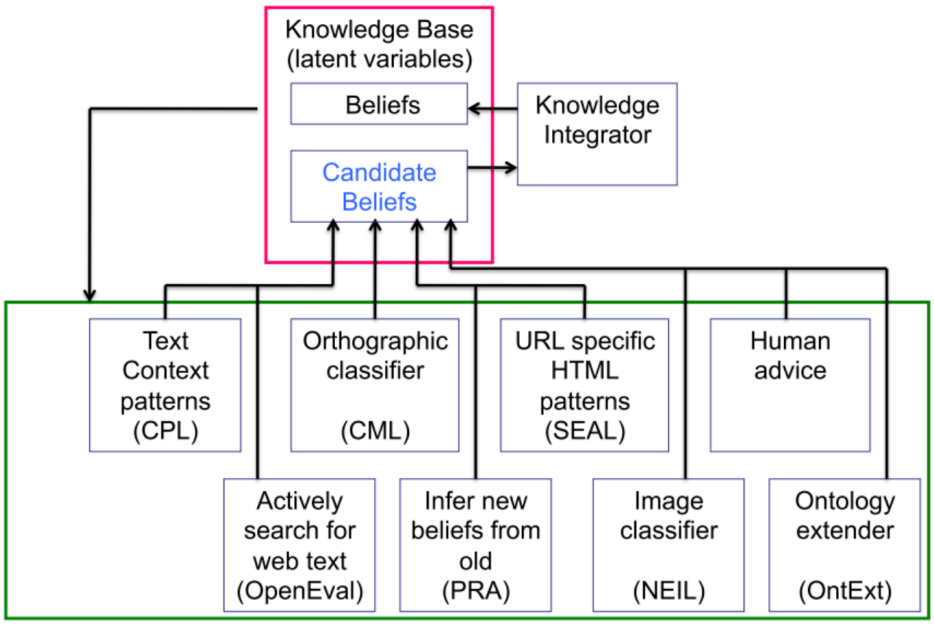

Figure 2.1: NELL's architecture. Information gathered by its modules is proposed as candidate beliefs with are then promoted (or not) to beliefs by the knowledge integrator. From Mitchell, T et al [Mitchell et al., 2015]

NELL has also been used as inspiration in many other areas such as face recognition [Subercaze and Gravier, 2014] and translation [Vieira and de Medeiros Caseli, 2014].

NELL's knowledge is organized into categories (cities, companies, etc.) and relationships between pairs of such categories such as "player plays for team". In order to gather such information NELL uses several extraction components which submit new instances to its knowledge base. These are then gathered and analyzed by a knowledge integrator that decides if they should be promoted to actual beliefs [Carlson et al., 2010a, Mitchell et al., 2015]. This process is detailed in Figure 2.1. The extraction modules use previously collected information from NELL's knowledge base to extract new knowledge that will, in the end, be provided (if accepted) as feedback for a new extraction iteration.

Like any lifelong learning system, NELL faces several obstacles such as data inconsistency and semantic drift causing labeling errors to accumulate. These obstacles have been described in bootstrap learning approaches [Curran et al., 2007] and can be mitigated by constraining the learning process. As has been stated, NELL uses several extraction components that can be forced to agree with each other. This coupled constraining process can be further exploited by inferring that instances can be mutually exclusive or part of larger supersets. Relations can be also typed checked to ensure that the instances of the proposed relations do in fact belong to the categories marked for those relations. Lastly, one can use Horn clause rules (from components like PRA) to further constrain the learning system [Mitchell et al., 2015].

In order to detail the workings of the mentioned modules one must first delve into one important field that has experienced exponential growth over the last few years. That field is Information

Extraction on the web [53]. Information extraction deals with adding structure to previously unstructured, or semi-structured, data. This can be done in any number of ways, but usually relies on patterns related to syntactic and semantic constraints on the position of entities within a text. Much like in knowledge extraction, information extraction makes use of several machine learning concepts. In fact, both areas share common ground and their integration has been shown to be mutually beneficial in areas like text mining [Nahm and Mooney, 2000].

In NELL, information extraction is done in a semi-supervised way known as bootstrapping where instances of a category are used as seeds to further extract new information about that same category [Wang and Cohen, 2009, Carlson et al., 2009]. In [Riloff et al., 1999] the authors describe a similar technique for creating dictionaries from a corpus of web pages

The very first step of information extraction deals, as expected, with information retrieval. In NELL this is done over a static corpus of 500 million web pages and internet queries on Google [Mitchell et al., 2015]. A common second step deals with transforming such information into structured knowledge. This can be done using Part of Speech (POS) techniques which divide sentences into known speech structures like nouns, verbs and adjectives or semantic features like subject, object and predicate. At this point, the content of the extracted information still suffers from a problem called co-reference. Co-reference relates to the fact that words can represent different concepts (for example the word "fired" can have many different meanings). In [Flach, 2012] the authors tackle this problem using a supervised learning approach. In NELL, however, this problem is resolved in a semi-supervised fashion by Concept Resolver [Krishnamurthy and Mitchell, 2011](this module will be referenced later in the report).

NELL components use, in one way or another, techniques from both information and knowledge extraction. The tasks of such components can be characterized as either Category classification, Relation classification, Entity Resolution or Inference Rules. These components are analyzed in Table 2.1. This table gives an overview of when the algorithms have been introduced to NELL and what kind of information is supplied and extracted by them. It then describes the model, that is to say the kind of patterns used to extract information, and what information is extracted and saved from each analysis. Finally, in the last two rows, one can see if these patterns are saved for reuse in future iterations (that is, if they are global or not), and considerations upon their main advantages on the overall system are laid.

2.2.1 NELL's components

The first analyzed component is Coupled Pattern Leaner (CPL). CPL uses co-occurrence statistics between noun phrases and contextual patterns to learn extraction patterns for each predicate of interest and then uses those patterns to find additional instances of each predicate [Carlson et al., 2009, 2010b].

The second algorithm, Coupled Set Expander for Any Language (CSEAL), uses wrappers of known instances to infer new instances surrounded by these same wrappers. While both these methods seem similar, in CPL we focus on textual patterns such as "X plays for Y" while in CSEAL we consider patterns that take the form of HTML code [Wang and Cohen, 2009, 2007].

It is exactly because of this that CSEAL is capable of extracting patterns from any language. It is also important to mention that while CPL extracts patterns from a static web corpus of web data, CSEAL uses Google queries to fetch information.

The third component in the table is the Coupled Morphological Learner (CML). CML classifies noun phrases based on morphological features such as affixes. One example of such features is the suffix "burgh" which is usually associated with city names like "Petersburg" or "Johannesburg". This component usually takes the role of a support algorithm by confirming facts proposed by other modules.

The fourth algorithm, Concept Resolver, does both sense induction and synonym resolution. Sense induction deals with creating pairs of instances and senses from the known instances that belong to more than one category (for example: "apple" belongs to both fruit and company and as such two senses should be created). Synonym Resolution on the other hand takes into consideration not only string similarity, but also relation patterns shared between instances [Krishnamurthy and Mitchell, 2011].

Path Ranking Algorithm (PRA) deals with extracting knowledge directly from the knowledge base using random walks to infer possible new beliefs [Gardner et al., 2013, Lao et al., 2011]. This algorithm largely replaced Rule Learner. Rule learner used Horn clauses to extract information much in the same way as done in FOIL [Quinlan and Cameron-Jones, 1993]. However, due to NELL's ever expanding knowledge base, this method became obsolete.

The sixth component, Ontology Extension (OntExt) was added to fill the gap of discovering new relations, something that NELL had been unable to do until then. To do so, OntExt first extracts sentences that contain instances of two categories. It then uses these sentences to build a context co-occurrence matrix to which K-means clustering is applied. The resulting clusters represent the new relationships to be proposed [Mohamed et al., 2011]. At the moment these need to be reviewed manually by a human expert before being introduced into the system.

OpenEval, much like CML, works mostly as a support algorithm for ensuring the precision of other systems. In a nutshell, OpenEval gathers seed instances of the involved category and uses them to query the web. It then extracts Context-Based instances and related keywords from the returned information in order to assess the truthfulness of the proposed fact [Samadi et al., 2013].

The eighth algorithm described in the table is Prophet. Prophet was proposed in 2011 but it is still being introduced into the system. It has, however, proved itself to be able to not only predict new relationships, but also identify those that were sometimes incorrectly added to the system. It does so by discovering open triangles over the already known relationships in NELL. The more open triangles found between two predicates the more evidence there is that these predicates should share a relationship. In a similar fashion, relationships which are unable to provide these same triangles should prove themselves to be incorrect [Appel and Hruschka Jr, 2011]. This process needs, however, to be monitored before acting on its own. It was this need that gave origin to NELL's conversing learning system. In order to verify its conclusions, Prophet turns to the wisdom of crowds. Using a Question and Answer system based on that of [Pedro and Hruschka Jr, 2012], Prophet is able to submit its prediction to a large number of users and assess its accuracy.

At the moment this process is done over Yahoo! Answers and Twitter (@cmunell) [Pedro et al., 2013].

The last component, Never Ending Image Learner (NEIL), has somewhat distanced itself from NELL and has a separate project website (http://neil-kb.com/). NEIL takes a different approach to NELL's information extraction techniques. Instead of web text NEIL focus its attention in visual imagery and the relationships observed between different objects identified in these images. Much like in the other components, relationships between instances can be forced to satisfy new relationships. In labeling instances of cars, for example, we expect wheels to be present in the extracted image since the relationship "Wheel is a part of Car" has been previously established. The semantic acquisition of images is done over text-based indexing tools such as Google Image Search [Chen et al., 2013].

	CPL [Carlson et al., 2009, 2010b]	CSEAL [Carlson et al., 2010b, Wang and Cohen, 2009, 2007]	CML	Concept Resolver [Krishnamurthy and Mitchell, 2011]	PRA [Gardner et al., 2013, Lao et al., 2011]	OntExt [Mohamed et al., 2011]	OpenEval [Samadi et al., 2013]	Prophet [Appel and Hruschka Jr, 2011, Pedro et al., 2013]	NEIL [Chen et al., 2013]
Cronology	2010	2010	2010	2011	2011	2011	2013	2011	2013
Input	Sentences (Tokenized by NPL)	Pure HTML	KB only[1]	KB only	KB only	Sentences (Tokenized by NPL)	KB only[1]	KB only	Visual Images
Extracts Category Instances	yes	yes	yes[2]	no	no	no	yes[2]	no	yes
Extracts Relation Instances	yes	yes	no	no	yes	yes[3]	yes[2]	yes	yes
Extracts new Relations	no	no	no	no	no	yes	no	no	no
Model	Lexical patterns	Wrappers from document being parsed	Morphological features	Relations Instances & Instance String	Relations Instances	Previously extracted patterns	Context-Based Instances and keywords	Relations Instances	Relationships between objects, scenes and attributes
Online Learning	New lexical patterns	—	New morphological features	Concepts	—	New relations patterns	—	—	New relationships between objects, scenes and attributes
Scope	Global	Local	Global	Global	Global	Global	Local	Global	Global
Main Focus / Advantages	General	Language independent	Improve precision of other algorithms	Word sense induction & Synonym resolution	Improve KB coverage	Extract new relations	Evaluate truthfulness of proposed instances	Identifies not only correct relations but also incorrect ones	Extract information from visual knowledge

Table 2.1: Characterization of NELLs extraction modules according to common features.

[1]The input for these components comes either from other components or from the AskNell system alone.
[2]The extraction of instances in theses cases should be seen as a confirmation of instances proposed by other components rather than by themselves.
[3]While Ontology Extension does propose new Relation Instances, it does so as a by-product of its main algorithm. These instances will act as seed for further extraction in the future.

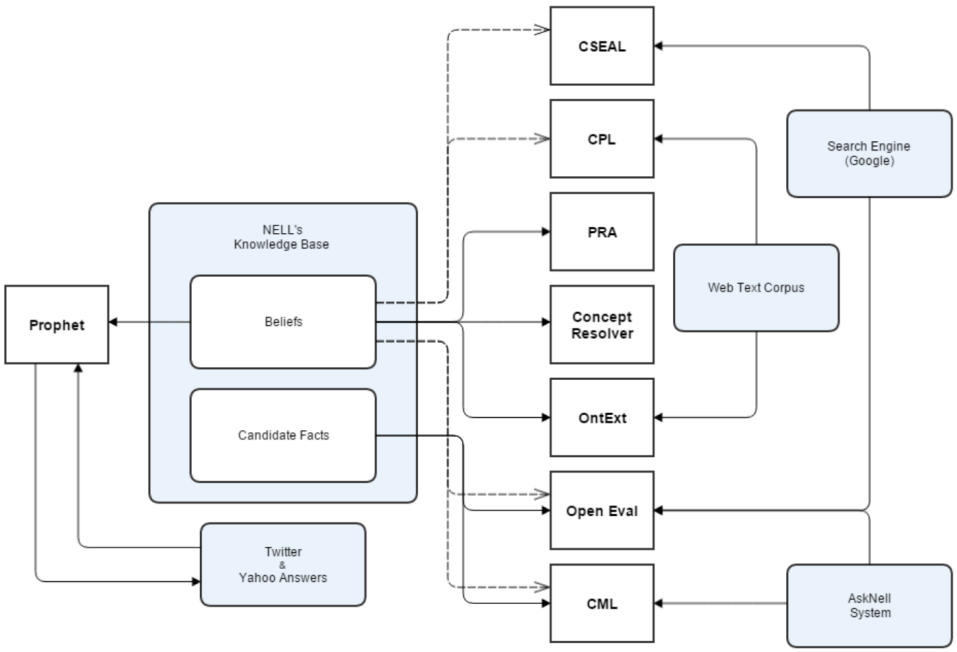

Figure 2.2: NELL's subsystem components input sources. The dotted arrows represent seed instances required by the algorithms while full arrows represent direct input from sources that can also include seed instances.

While not explicitly declared in Table 2.1 these algorithm take advantage of several external input sources. These are described in Figure 2.2. The dotted arrows represent seed instances required by the algorithms while full arrows represent direct input from sources that can also include seed instances. As discussed in [Mitchell et al., 2015], NELL still lacks a mechanism to monitor its performance and progress over different categories. At the same time, different modules perform better or worse for each category. Due to the amount of data present in NELL's knowledge base, monitoring the performance of different modules over different categories should be no easy feat. In the next section we will discuss Metalearning, a new field of machine learning that might provide an answer to this problem.

2.3 Metalearning for Algorithm Selection

As seen in the last chapter, the enormous amount of data collected by some machine learning applications can hinder the choice of algorithms that are best suited to deal with a learning task. One possible way of tackle this problem would be to test a small number of different algorithms, which would imply a considerably high computing time, as well as the presence of a skilled expert to hand pick the set of algorithms to be tested [Brachman and Anand, 1996]. Metalearning comes as a solution by steering the user in the algorithm selection task while taking into account the domains of application [Smith-Miles, 2008].

In order to do so, metalearning focus its attention on the underlying qualities of the learning process. This is usually described as metaknowledge. Metaknowledge can contain information about not only the performance of algorithms over datasets similar to the one under analysis, but also about models that characterize such algorithms and metrics to compute similarity between tasks or datasets. It is thus possible to describe metalearning as

> "...the study of principled methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes." [Brazdil et al., 2008]

The process of algorithm selection can also be expanded to cover operations (such as discretization or thresholding) that must be done before and/or after the use of such algorithms. In this case, the problem is viewed as a selection of plans that contain the sequence of events rather than the algorithm by itself. In a similar fashion this can be applied to model combination for problems such as boosting [Chan and Stolfo, 1993].

In the process of algorithm recommendation, the role of metalearning is to generate metaknowledge that can map the characteristics of the problem at hand with the expected performance of algorithms over this same problem. By doing this we expect to minimize the number of alternative algorithms to be tested without compromising the quality of the results. One can further improve this process by ranking the possible algorithms accordingly to their expected performance. This can be done by using machine learning algorithms such as k-Nearest Neighbors [Brazdil et al., 2001]. We start by selecting a set of datasets that are similar to the one being analyzed and then combine their target values to generate a prediction for the new example (dataset). Since the performance of different algorithms for these datasets is known (they have been previously computed), one can extrapolate that a similar performance should be accounted for this new example. It is also clear that this performance can be used to rank different algorithms instead of just naming the best one. Other algorithms beside k-Nearest Neighbors can be used for algorithm recommendation. In [Alexandros and Melanie, 2001], for example, decision tree-based algorithms are considered. In regression the problem is somewhat more complex and one has to generate as many metamodels as there are algorithms in order to correctly estimate their expected performance. The algorithms for metalearning described so far imply that the extractable metaknowledge is propositional. This might not hold for many problems. FOIL [Quinlan and Cameron-Jones, 1993], a system that is also explored in NELL, manages to exploit nonpropositional descriptions of the datasets using inductive logic programming. The choice of what algorithm to use in meta-level learning needs to consider not only the task in hand (that is if the approach matches the problem), but also the evaluation measures used to differentiate separate ranking algorithms [Brazdil et al., 2008].

Ranking accuracy can be estimated using Spearman's rank correlation coefficient [Spearman, 1904]. One must, however, compare this measure with an appropriate baseline in order to understand if the model is useful or not. A commonly used baseline for this type of problems is the default ranking. This baseline consists of the averaging of all the target rankings considered.

Some other methods have been proposed such as the Success Rate Ratios and Significant Wins [Brazdil and Soares, 2000]. It is important to note that just comparing a ranking method with another or even with its baseline is not sufficient. In order to properly compare ranking methods there needs to be a statistical significance between them. This can be done using a combination of Friedman's test and Dunn's multiple comparison technique as in [Brazdil et al., 2003]. The evaluation described suffers from the fact that we are comparing every base-algorithm ranked. In most situations it is more useful to only consider those algorithms that have the best score. This is called Top-N evaluation where N is the number of algorithms (to be considered) with the best overall ranking. It is also important to consider multiple performance metrics. Execution time can be a very important factor when deciding between algorithms. In fact, in some cases, one could sacrifice accuracy in order to lower execution time. In the end, it is up to the user to define the importance of each metric in the final ranking of each algorithm. It is also useful to consider the worst-case performance of a ranking method. That is to say that in a top-N solution, only the dataset with the lowest accuracy should be considered rather than the average of the N datasets [Brazdil et al., 2008].

The ranking of algorithms needs, at the same time, to take into account the choice of parameters for each algorithm. This, in itself, can be considered another ranking problem. In fact, if we consider each parameter configuration as a separate algorithm, then the methods described above can be applied in the very same way [Soares and Brazdil, 2006]. Pairing algorithm recommendation with parameter settings recommendation implies that the amount of metaknowledge to be computed grows exponentially.

One must thus devise a strategy for picking the appropriate base-algorithms and their respective parameter configurations. In order to select base-algorithms one has to consider both availability (that is if the user has access to implementations of this algorithm) and applicability. This last factor deals with the goal of the problem itself. That is to say that, if, for example, we want to predict a continuous variable then only regression algorithms should be considered. We can further constrain the set of available algorithms by taking into account their assumptions over the considered dataset. Naive Bayes, for example, assumes that variables are independent [Rish, 2001]. One can exclude this method by proving a correlation between variables. It is also useful to perform heuristic experiments on the set of base-algorithms so as to make sure that they meet some basic standards. The algorithms must be both relevant (must perform better than the baseline) and competitive (expanding the set of parameter settings brings no significant improvement). Individually, there cannot be a set of parameters such that one setting performs better in every dataset and, at the same time, every setting must be the best alternative for at least one dataset [Brazdil et al., 2008].

It is also important to consider the actual format of the provided algorithm ranking. This can take the shape of a single algorithm (the best one) or a subset of the best algorithms (according to a performance threshold). Alternatively, one can consider sets of rankings that can be either linear and complete (when there are no ties between algorithms), weak and complete (when there are ties or performances that are not significantly different) and linear and incomplete (when there is

not enough data to correctly identify the algorithm performance). These types of ranking can be visually represented using Hasse diagrams [Pavan and Todeschini, 2004].

While the relative ranking of algorithms is informative, there is a need to understand the actual performance of each algorithm rather than its relative performance over others. In order to do this, one needs to turn the task of recommendation into one of regression where each base-algorithm is represented by a regression model. The performance outputted by such methods still needs some context. While a high accuracy is hard to achieve in some datasets, in others it might be trivial. In order to contextualize the performance one much turn to relative measures such as the distance of performance to the best algorithm or baseline, or simply normalize the obtained performances [Brazdil et al., 2008].

The whole process of algorithm ranking in metalearning is dependent on the data characteristics (metafeatures) of the datasets involved. These features can take many forms, but must also follow some patterns. First, they must be discriminative, that is to say that they should contain information that distinguishes the dataset between base-algorithms. This is particularly challenging when the base-algorithms represent the same algorithm with different parameter settings. In these cases algorithm-specific metafeatures might be needed [Soares and Brazdil, 2006]. The metafeatures should also not be computationally heavy, otherwise the savings from using metalearning become redundant. Lastly, the number of metafeatures should not be too large when compared to the amount of available metadata so as to prevent overfitting. The characterization of a dataset needs not to be related to its actual features. Information about the base-algorithms used might also be used to generate metafeatures such as the type of data they are able to deal with or the sensitivity to irrelevant attributes to name a few [Brazdil et al., 2008].

2.3.1 Data Characterization and Data quality

In the context of metalearning there are three different approaches to data characterization. The most common one is to use simple, statistical and information-theoretic measures. Simple measures consist of easy to calculate descriptive quantities such as the number of examples or number of features in a dataset [Rendell et al., 1987]. Statistical measures include properties like mean skewness of numeric features and information theory provides measures such as class entropy. A second type of approach to data characterization is to induce metafeatures from models that fit the data [Hilario and Kalousis, 2001]. A simple example would be to analyze the morphological properties of a decision-tree (such as number of leaf nodes) that models the dataset. Yet another approach is the use of landmarkers [Bensusan and Giraud-Carrier, 2000]. Landmarkers are estimates of algorithm performances which can be obtained by either running simplified versions of the algorithms or subsampling the data. As has been stated, measures that identify correlations between features can also be quite useful in that they constrains the subset of base-algorithms that can be selected.

One issue with metalearning is the lack metadata available. Most of the data involved in machine learning are either of private domain or simply not published. While there are some repositories of datasets available online, such as the one at the University of California at Irvine

(UCI), the amount of information needed for a metalearning study is simply not available nor easily accessible. Some have proposed a technique of generating synthetic datasets [Vanschoren and Blockeel, 2006] that mimic natural data. Others have proposed the creation of new datasets by manipulating existing ones by changing the distribution of data or the structure of the problem [Hilario and Kalousis, 2000]. In some areas like datastreams and extreme data mining [23], the number of meta-examples are so large that we can simply subsample the data to create different datasets and avoid this issue.

Lastly, it is important to reflect on the quality of metadata obtained. The first problem when dealing with datasets relates to their representativeness. Some argue, for example, that because datasets within the UCI repository have been heavily preprocessed, the data no longer represents a real world problem [Saitta and Neri, 1998]. Another issue of metadata quality involves missing or unreliable performance data. In order to perform metalearning over a set of datasets, information about the performance of the selected base-algorithms over these datasets needs to be readily available. This also means that, when adding a new dataset, one needs to compute the performance of each base-algorithm over that same dataset. The same can be said when adding a new base-algorithm. In this case we need to compute its performance over the pool of all available datasets. There is also the problem of not being able to compute an algorithm's performance over a dataset. In this case, one approach is to assign a baseline performance such as "predicting the most frequent class" (in a classification problem). Finally, the values of metafeatures may also be missing or simply impossible to compute. This is the case of mean skewness which is not applicable if there are no numeric features over the dataset [Brazdil et al., 2008].

In [Kadlec and Gabrys, 2009] the authors describe a lifelong learning architecture that successfully applies metalearning concepts to improve its performance. In the next section a previous approach to metalearning in NELL will be described.

2.4 Previous work

In [dos Santos, 2014], a possible implementation of metalearning in NELL was described. In this study the author explored the idea of using metalearning for algorithm prediction and focused his attention on a single component, CML. The main objective of this work was to predict the performance of CML over different categories using metalearning. In order to do this, several algorithms were used such as Recursive Partitioning and Regression Trees, SVM, Neural Networks and Partial Least Squares Regression. A preprocessing step was needed to reduce the dimensionality of the received data. This was done using Principal Component Analysis. While not entirely successful, this work represents a good first step into the realms of metalearning in an unusual context such as NELL's framework.

Chapter 3

Metalearning approach for self-reflection in NELL

As mentioned before, NELL still lacks the ability of self-reflection. That is to say that, at any given time, NELL is unable to detect those categories at which it is learning less or worse. Indeed, in some categories such as zipcode, it is possible that NELL simply won't be able to learn anything new from some point on because there is nothing more to learn about the subject. The ability to detect these outliers would allow NELL to focus on the categories that it finds most desirable. While not the main focus of this report, we suggest three ways on how this could be implemented on the system:

Firstly, when a category seems to be under-performing (on one or more components) we could downgrade the score of the instances proposed by the components (marked as "p" in Figure 3.1) for this particular category so that they are less likely to be promoted by the system (specifically by the Knowledge Integrator).

Secondly, the system could also detect whether the performance of categories has been declining over time and launch an alert. This alert could then be interpreted by NELL and, if needed, introduced to human interaction over the conversing learning system described in section 2.2. Finally, NELL could simply ignore the under-performing categories for a time period and only extract new information on the remaining categories reducing the computational time for each iteration.

One might also propose that, because we can estimate the overall performance of each component, we could value the proposed facts by one component more highly than those proposed by another. Due to the non-competitive nature of the components this is probably not a good idea. As already mentioned the components not only extract information from different sources, but also do so in a very different manner and should be seen as complementary rather than competitive.

In this section we will describe a metalearning approach to solve this self-reflection gap on NELL's learning system. We will start by describing the model behind our solution and how it

was created.

3.1 The Model

As pictured in Figure 3.1, we start out by creating a metamodel. This is done by computing metafeatures from the set of proposed instances from previous iterations. Since we know how many of these instances have been promoted to beliefs by the Knowledge Integrator, we can pair these metafeatures with their acceptance ratio and create a metamodel that relates the two. For each component, the source of information can be completely different (as discussed in Figure 2.2) and is thus simply represented as "Data Source".

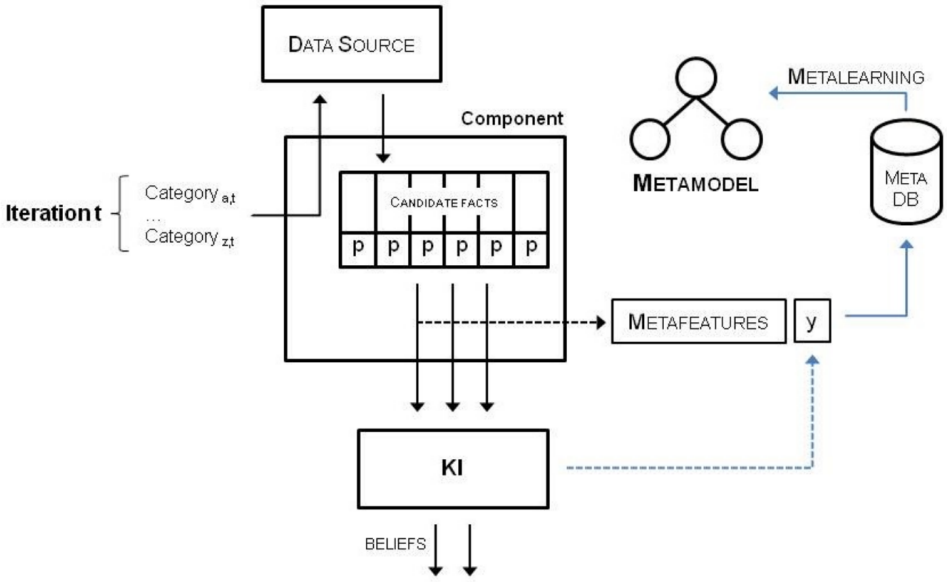

Figure 3.1: Creation of the Metamodel. Proposed facts are introduced by each component and metafeatures extracted. These are then paired up with the acceptance ratio(y) from the Knowledge Integrator and a meta database is created. This database can be then used to create a metamodel.

From then on, as seen in Figure 3.2, at any moment, we can compute the metafeatures of the current proposed facts and predict how many of these will be accepted before they are passed on to the Knowledge Integrator.

3.2 Data Preparation

The candidate facts represented in the model were not directly available to us and had to be heavily parsed from the documents available on NELL's project page at http://rtw.ml.cmu.edu/rtw/resources. At the moment of parsing the document contained the first 890 iterations ran by NELL. These documents provided all instances proposed by every algorithm over all iterations (initial header represented in Figure 3.3). The first step was to separate instances proposed by

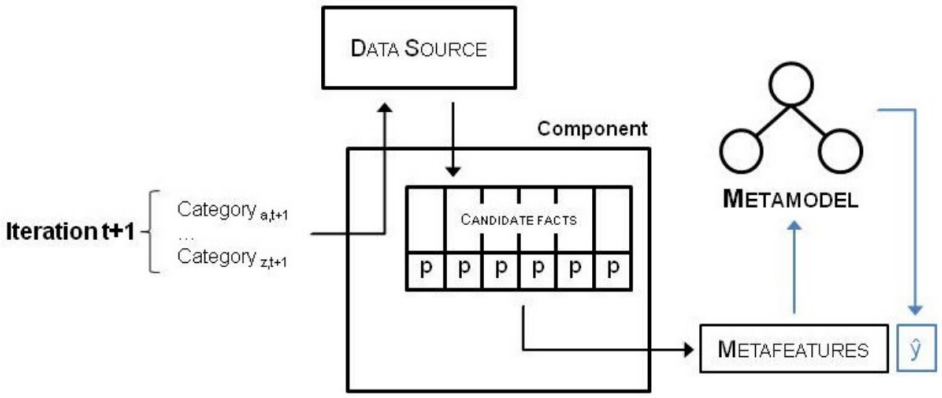

Figure 3.2: Newly available data is fed into the previously created metamodel so as to predict how many instances will be accepted in the future($\hat{y}$).

different components and then filter them by category. Later, these instances were further divided by iteration.

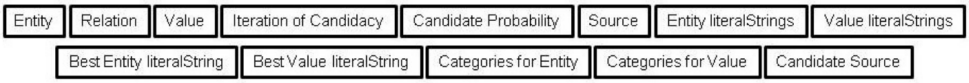

Figure 3.3: Header of the initial file containing every proposed instance.

Unfortunately, this file did not provide information over whether the instances had been promoted or not. A separate file with a similar structure provided this information and cross-checking had to be performed.

Since some of the information provided by the file was either repeated or unhelpful, the selected fields were reduced and summarized into the following structure:

Figure 3.4: Header of a typical file describing instances proposed by a component for a category in a single iteration.

A typical example of this kind of structure could be (ignoring parenthesis):

> *University of Porto* (Primary Instance), *University* (Primary Category), *12* (Proposed In), *15* (Accepted In), *CPL* (Proposed By), *0.95* (Probability), *relation* (Type), *universityincity* (Relation Name), *Porto* (Secondary Instance), *City* (Secondary Category)

Since these files are already divided by component (ex: CPL, SEAL, etc.) one might question the existence of the field "Proposed By". However, one instance can be proposed by more than one component and at different iterations. In order to solve this problem, in those cases, fields like "Proposed In", "Accepted In" and "Proposed By" were allowed to have multiple values separated by commas.

While not obvious in the initial file, the now parsed information clearly separates the two types of instances present on NELL's KB: category instances and relation instances.

This last notation not only contains more information but is also more easily parsable and intuitive. In the event of an instance of type category, the last 3 fields will appear as empty. It is also important to note that the field probability represents not the actual probability of promoting the instance, but the score given by the component to that same instance. This score is calculated independently by each component. In CPL, for example, the score is proportional to the number of times an instance is found by the algorithm on the web corpus.

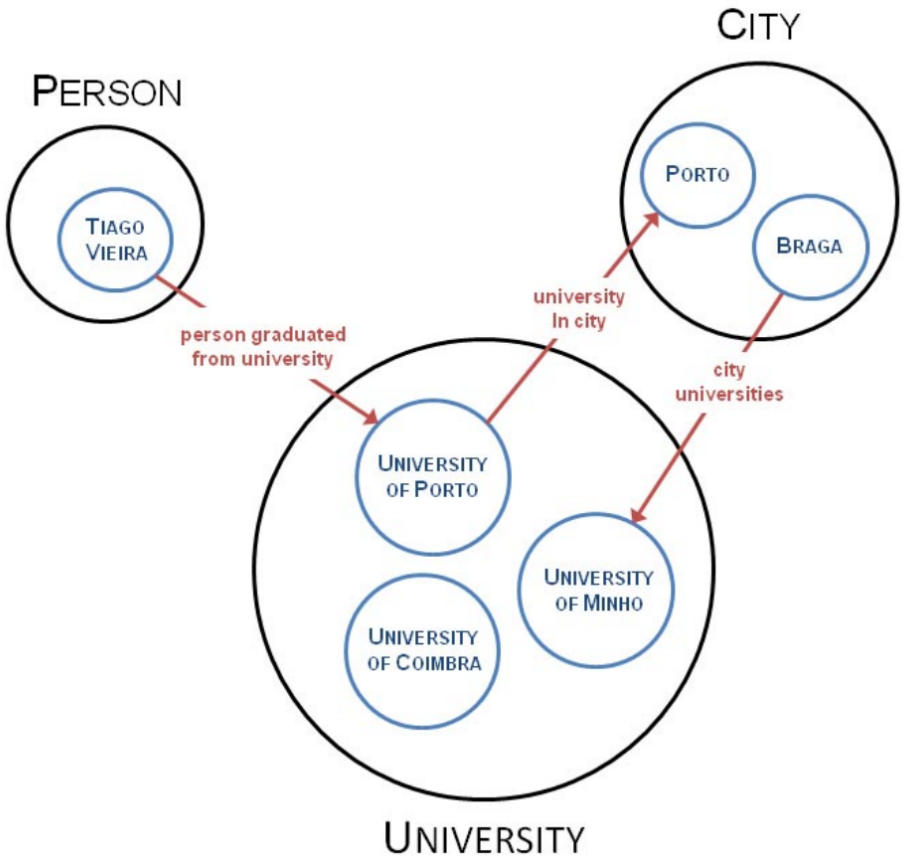

Figure 3.5: An example of a possible set of proposed instances related to category University. The blue circles represent category type instances while the red arrows constitute relation type instances.

Since we are dividing instances into different categories, some redundancy had to be introduced. In the case of instances representing relations (between categories) these had to be placed

in both categories involved, doubling the number of relation instances. In the example shown in Figure 3.5 both "university in city" and "city universities" would count as relation instances in University and City.

3.3 Metafeatures

From the fields described above, one can distill several metafeatures. These can be loosely divided into 3 categories: General, Involving Relations and Activity. In the final dataset each line will correspond to a set of metafeatures summarizing one iteration worth of instances of one category as described in Figure 3.1.

3.3.1 General Metafeatures

The metafeatures in this category relate to common fields between category and relation type instances.

Number of Instances

Represents the number of proposed instances on an iteration for a category. Different behaviours might be expected from different ranges of number of instances.

Percentage of Category Instances

Represents the percentage of category type instances over all proposed instances. The percentage of relation instances will be thus redundant.

Average Probability & deviation

Represents the average score given to instances by the component. Since the Knowledge Integrator takes this into account when promoting instances into beliefs this variable might be extremely important. It should be noted that a high score on this variable does not guarantee that instances are promoted, since they can still fail typechecking and mutual exclusivity constrains created by the system.

Category and Relation probability & deviations

Represents the average score for instances of different types and their corresponding standard deviations.

3.3.2 Relation oriented Metafeatures

The metafeatures in this category rely on the graph like structure of NELL's relation type instances.

Number of Mutual Exclusive Relations

Represents the number of mutual exclusive relations between this category and others. This is the only static variable in the metafeature set. Since the number of possible related categories is inversely proportional to this value, this might affect the overall relevance of other relation oriented metafeatures

Number of relations with other categories

Represents the number of relations with other categories. If there exists at least one relation type instance between categories then those categories are related. If however there is more than one relation type instance relating two categories this will not increase the number of relations between these categories.

Input & Output relations with other categories

Since relations are directional (the relation "athleteplayssport" relates "athlete" with "sport" but not the other way around) we can divide the number of relations into those which relate other categories with this (input) and those with which this category relates to others (output).

Total number of unique relations

Represents the total number of individual relations proposed on this iteration. If one or more instances share the same relation then we only account for one relation. Having this variable as well as the number of relations with other categories means that one can determine the average number of instances that relate the analyzed category with others.

Average Number of unique Input/Output relations & deviations

Separately evaluates the average number of unique relations for input and output in the same way as described above and also scores their deviations for each particular scenario.

3.3.3 Activity related Metafeatures

The metafeatures in this category focus on the behaviour of past iterations and the activity of other components so far.

Average number of repeated instances

Represents the average number of instances proposed in this iteration that have already been proposed in the past by this or other components.

Average number of already accepted instances

Represents the average number of instances in this iteration that have already been accepted before. Sometimes an instance is proposed even though it has already been accepted, by this or another component, in the past.

Average number of instances proposed and accepted in the last 14 iterations

Represents the instances that have been proposed and accepted in the last 14 iterations. If a component is actively proposing instances that are immediately accepted, then there might be a higher chance that those proposed in this iteration might also be accepted. There is also the chance that its instances are never accepted.

Percentage of iterations with no proposed instances in the last 14 iterations

Represents those iterations where no instances have been proposed. This might point out

that the system has been unable to find new facts (maybe because there are none left to be found) and thus that instances have a lower chance of being accepted.

Average number of instances proposed on past iterations

Represents the average number of instances the component proposes on each iteration for the considered category. A sudden increase or decrease of the number of instances proposed might mean that the algorithm is being less or more specific, which can directly affect the chance of its instances being promoted.

Average number of instances proposed on the last 14 iterations

While similar to the last metafeature, this opens the possibility to detect local anomalous behavior and correct the assumptions made before.

Average number of category/relation instances proposed on past iterations

Represents exactly the same information has the last two metafeatures but for each type of instance considered (making it a total of 4 metafeatures).

Decrease/Increase of the number of instances proposed on the last 14 iterations

Represents the slope of the linear regression of the last two iterations that have proposed instances. If, in the last 14 iterations, no instances have been proposed in more than one iteration, then this takes the value of zero.

Decrease/Increase of the number of category/relation instances proposed on the last 14 iterations

Represents exactly the same information as the last metafeature but for each type of instance considered (making it a total of 2 metafeatures).

Ratio of proposed instances on this iteration over all categories

Represents the ratio of proposed instances by one category over the number of proposed instances for all categories in one iteration.

Ratio of proposed category/relation instances on this iteration over all categories

Represents exactly the same information as the last metafeature but for each type of instance considered (making it a total of 2 metafeatures).

The metafeatures described above could be further explored. In the case of relation oriented metafeatures a multitude of graph related features could still be computed. However, since relation instances account for only a share of the total instances proposed by each algorithm, these features might not bring anything new to the table. Indeed, some components like CML only propose instances of type category. In fact, in this case, those metafeatures must simply be ignored.

In the next chapter, we will explore in greater detail the data and the exploratory analysis performed. We will then discuss the two experiments performed over different sets of metafeatures described in this chapter. Lastly, we will evaluate the results of such experiments and discuss their importance and implications on NELL.

Chapter 4

Implementation and Results

In this chapter, we will discuss the details of implementation of the designed solution. We will start by describing the exploratory analysis made over the available data and the consequences of such analysis. We will then describe the experimental methodology and the two experiments performed over different metafeatures. Finally, we will discuss the results obtained on this two experiments and what future work could be done.

4.1 Exploratory Data Analysis

To have a feel of the amount of information provided by the individual components, the first step was to plot the number of proposed instances by each of them. In Figure 4.1 we can see that CPL produces much more instances than every other component. This is due, in part, to a restructuring of its algorithm that we will discuss later. Other components like Ontology Extension, PRA and OE have proposed less instances simply because they were added to the system later than the other components. It is also worth noting that components like CMC and OE, that take a more support like role in the system, would necessarily have proposed less instances than SEAL or CPL since they only confirm/support the instances proposed by them.

While analysing the number of accepted instances (Figure 4.2) it is worth to point out that, while CPL produces much more instances than SEAL, the instances it proposes have a much lower chance of being accepted. As expected, CMC could not produce more accepted instanced than any other component.

In Figure 4.3, the behaviour discussed before is easier to discern. The observed behaviour of CMC was expected since it defaults to the role of confirming instances proposed by other components.

The behaviours we just described, however, vary with time. Figure 4.4 illustrates the number of proposed instances of each algorithm over the 890 iterations.

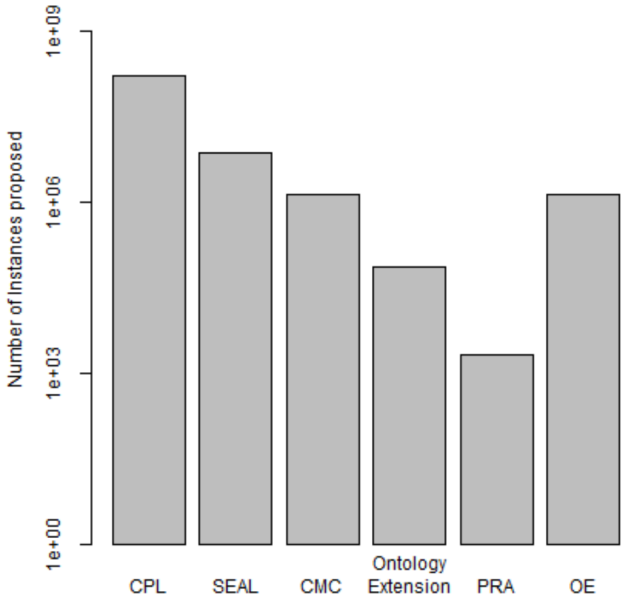

Figure 4.1: Number of proposed instances by every component over the first 890 iterations of NELL's run.

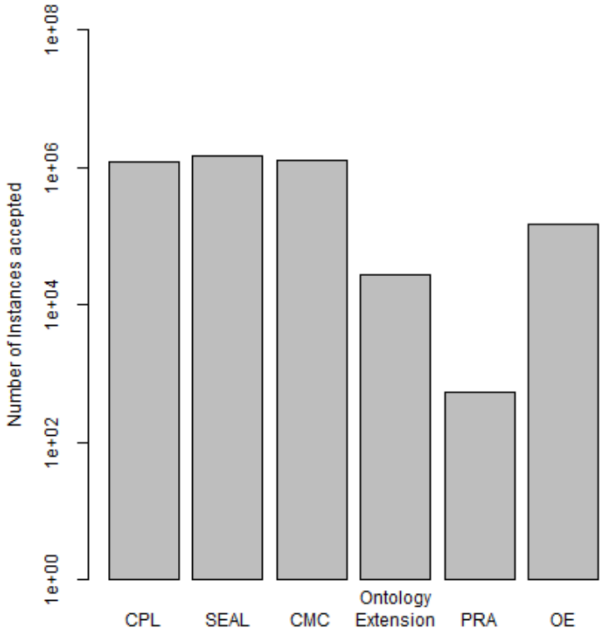

Figure 4.2: Number of accepted instances by every component over the first 890 iterations of NELL's run.

It is clear that some odd behaviours occur, which can, in most part, be explained from system changes that occurred at the time. At iteration 490, for example, the system suffered a redesign to deal with polysemy and the number of proposed instances at the time raised in more than one

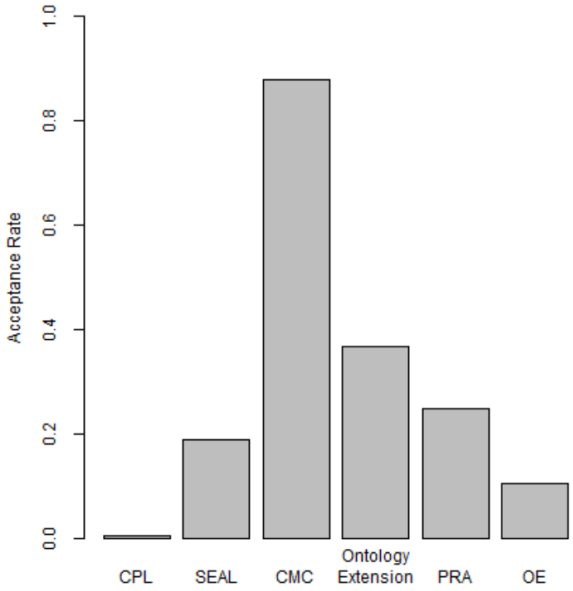

Figure 4.3: Acceptance rate of instances proposed by every component over the first 890 iterations of NELL's run.

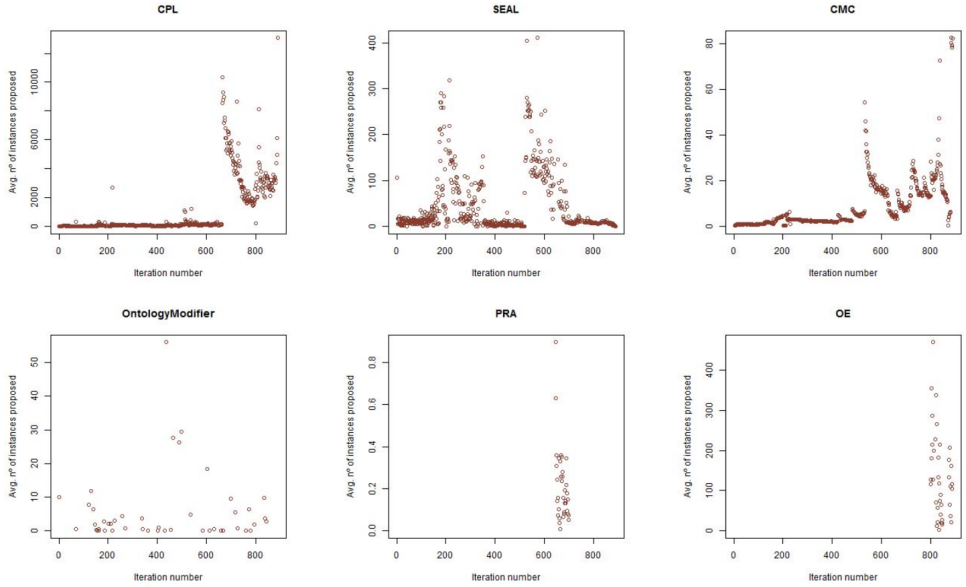

Figure 4.4: Average number of instances proposed by category in each iteration over 890 iterations.

component. At the same time CPL began to be able to resubmit previous instances, which led to an increase of proposed instances. On the other hand, at around iterations 660, many instances were deleted thus removing many constraints that held back CPL from proposing instances. At the same time, new predicates and seeds were added to the system at around iteration 700 and 729. At

iteration 724 and 799 some alterations related to case-sensitivity and internal scoring were made in CPL, improving its overall performance. These changes forced CPL to re-learn and re-propose instances already learned. In SEAL's case, some of the observed spikes have a more practical nature. At iteration 200 some new predicates and seeds were added due to a competition where NELL was participating. At the same time, the whole system started to allow different concepts for one same instance (prior to this point, NELL would not allow "apple" to be both a company and a fruit even thought it had the ability to differentiate between the two). From there on, there was a big drop in proposed instances which corresponds to a period of inactivity related to a problem with the Google query system (which ended up being switched to Bing). The high values observed in the CMC plot are related to the spikes of activity of CPL and also to the introduction of a newer version of CMC at around 837. Lastly, the outliers in Ontology Modifier correspond to the addition of predicates and seeds for various competitions such as the TAC KBP competition at iteration 425 and 830-84.

By analysing how many instances were accepted in each iteration (Figure 4.5) it is easy to see that the burst of proposed instances by CPL and others, around iteration 650, leads to a higher number of instances being promoted not only at CPL but also in other components.

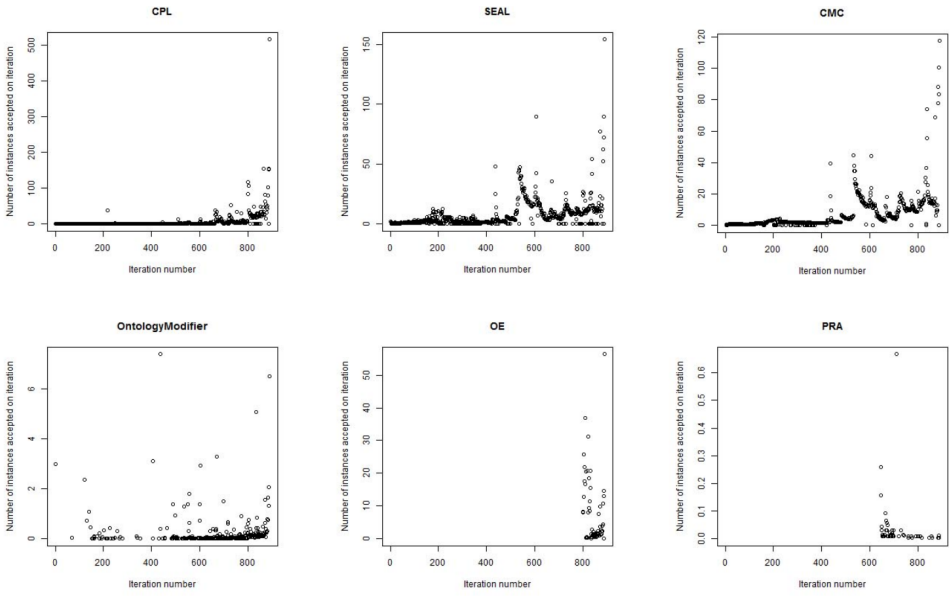

Figure 4.5: Average number of instances accepted by category at each iteration over 890 iterations

This effect is particularly clear when plotting how many iteration it takes for an instance to be promoted over time (Figure 4.6).

In Figure 4.6 we plotted the third percentile of the number of iterations that it takes for an instance to be promoted. The third percentile was again scored to average the whole space of categories. The red line represents the worst case scenario (that in which every instance took the maximum number of iterations, in our time period, to be promoted - if an instance has been

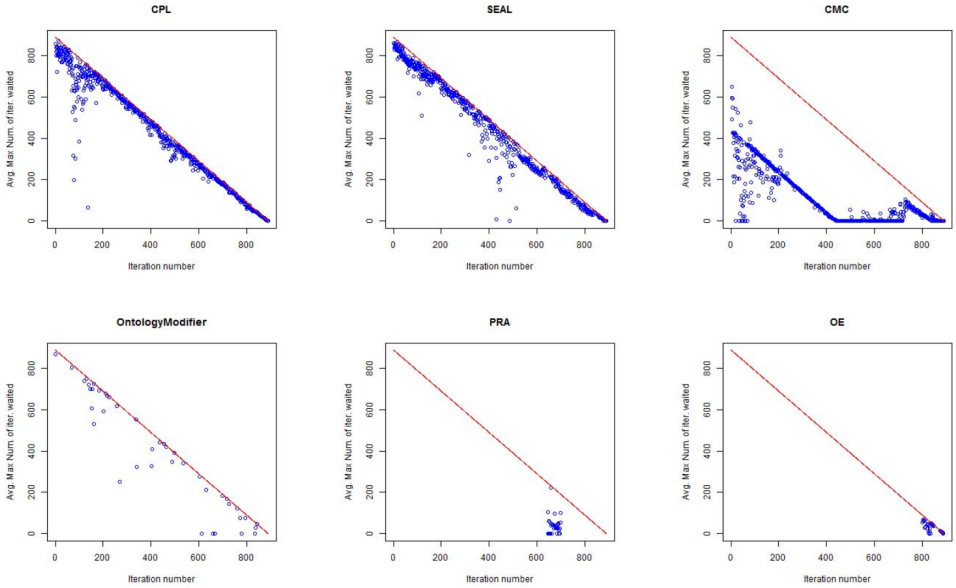

Figure 4.6: Third percentile of the third percentile of iterations waited by an instance to be promoted by category over each iteration for the totality of NELL's run.

promoted, then the number of iteration this instance had would have to wait would necessarily have to be less than 890 minus whatever iteration we are in).

While it is not extremely clear why we observe such high waiting periods, when reducing our timeframe to the first 500 iterations we face a completely different scenario (4.7).

In this new timeframe it is clear that instances took a lot less time to be promoted and a maximum waiting time can be deduced. In order to fit most components, we proposed that in the first 250 iterations, instances take at most 250 iterations to be promoted (if they don't then they will probably never be promoted). From here on when referring to an instance being promoted we are referring to it being promoted in the next 250 iterations.

4.2 Experimental Methodology

As mentioned before, we reduced our data to the first 500 iterations of NELL's run. From the exploratory analysis, we found that in the first 250 iterations most instances promoted are so in an interval of around 250 iterations. In our analysis, we will thus focus our attention in the first 250 iterations so that we can be sure that instances have enough time to be promoted. In order to test the performance of our models on unknown data we will further divide this 250 iterations into a training set and a testing set as described in Figure 4.8.

As described in the last chapter, we performed two separate experiments with different sets of metafeatures. Our first experiment focused on common variables between components such as number of instances proposed or ratios of category and relation type instances. The metafeatures used in this experiment are listed bellow.

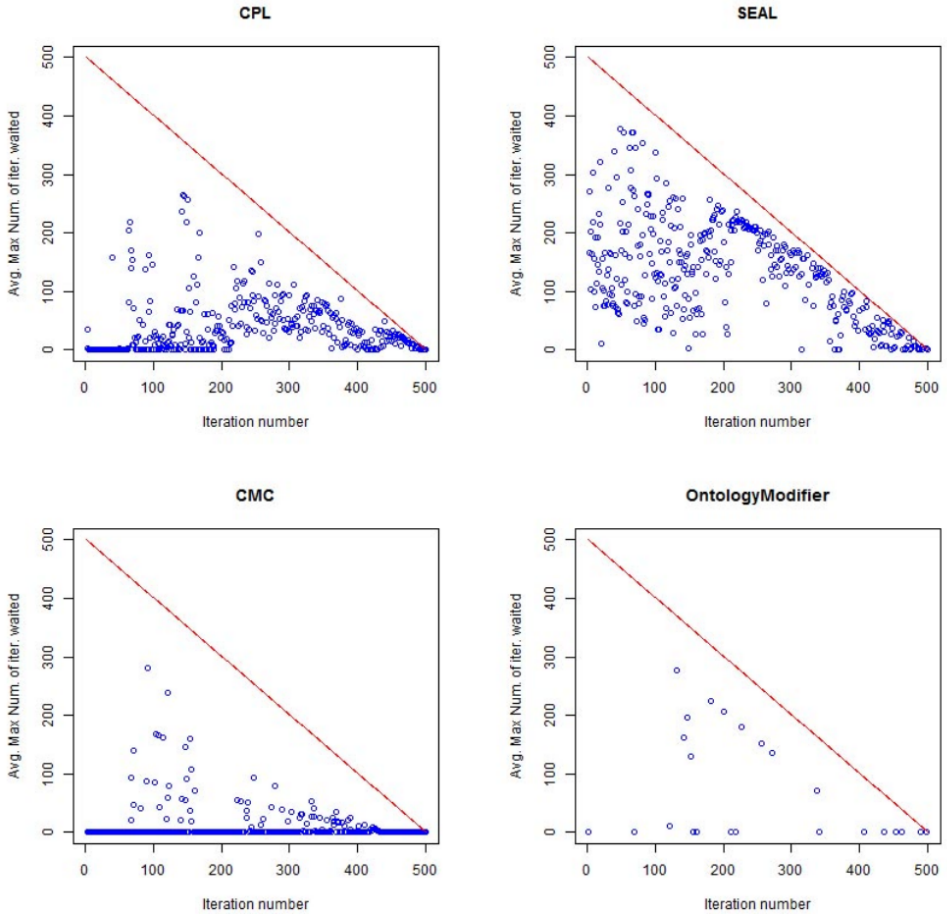

Figure 4.7: Third percentile of the third percentile of iterations waited by an instance to be promoted by category over each iteration for the first 500 iterations. Since both PRA and OE have been added after iteration 500, they are not represented in this figure.

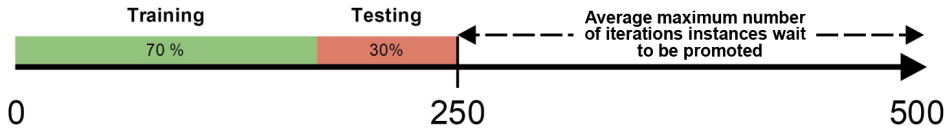

Figure 4.8: Time frame of our experiments. Considering only the first 500 iterations of NELL we consider only the first 250 iterations and divide them into training(70%) and testing(30%).

- Number of Instances
- Percentage of Category Instances
- Average Probability & deviation
- Category and Relation probability & deviations
- Number of Mutual Exclusive Relations

- Number of relations with other categories
- Input & Output relations with other categories
- Total number of unique relations
- Average number of unique Input/Output relations & deviations
- Average number of repeated instances
- Average number of already accepted instances

Our second experiment considered all the above metafeatures plus those related with the activity of the component beforehand. Those additional metafeatures are listed bellow.

- Average number of instances proposed and accepted in the last 14 iterations
- Percentage of iterations with no proposed instances in the last 14 iterations
- Percentage of iterations with no accepted instances in the last 14 iterations
- Average number of instances proposed on iterations (so far)
- Average number of instances proposed on the last 14 iterations
- Average number of category/relation instances proposed on iterations
- Decrease/Increase of the number of instances proposed on the last 14 iterations
- Decrease/Increase of the number of category/relation instances proposed on the last 14 iterations
- Ratio of proposed instances on this iteration
- Ratio of proposed category/relation instances on this iteration

The last step was to run several well known machine learning algorithms over the described dataset and evaluate their performance. Since we are facing a regression problem the chosen measure was RMSE. As described in 2.1, this error, by itself, does not allow us to evaluate the effectiveness of our models. The mean and median of the training set were thus used as base algorithms and compared to our models using the formula:

$$RRMSE = \frac{RMSE_{model}}{RMSE_{base}}$$

We further explored the performance of our models over each category using as base model the mean and median of that same category on its training data. Those categories that did not propose instances in the training set, but did so in the test set, were discarded (<0.1%). Until now, we were giving the same importance to categories with different amounts of instances proposed.

However, since we are predicting the acceptance rate of each iteration, it is important to factor in the number of instances proposed at each iteration (after all, predicting that 70% of instances will be promoted for a small number of instances is very different than doing so for a large number of instances). In order to take this into account, a weighted averaged was added to the evaluation procedure.

The machine learning algorithms used in this work were chosen not only for their remarkably good predictive capability in other areas, but also for their availability, ease of use and overall popularity. They are:

Multiple Linear Regression

Multiple Linear Regression is an extension of linear regression where more than two attributes are used and the data is made to fit a multidimensional surface. This method has the advantage of being fast and easily interpretable.

SVM

SVM, or Support Vector Machine, is a more elaborate method that uses nonlinear mapping to transform the data into higher dimensions. It then searches for a linear hyperplane that separates classes in this new dimension. This hyperplane uses support vectors (tuples) and has a variable margin that allows him to deal with overfitting to a certain extent. Although slower than the last method, SVM is able to model complex non-linear problems.

KNN (with and without feature selection)

KNN, or K Nearest-neighbors, is a machine learning algorithm that creates a n-dimensional space (where n is the number of attributes of our dataset) and uses it to find the nearest neighbors of unknown tuples. Using these known nearest neighbors the algorithm then extrapolates the target value for the new tuple. Since the distance function used in determining the nearest neighbors uses every feature available, those features that are irrelevant will factor in as much as those who are not. In order to avoid this issue, RRelieF was applied prior to its use.

Neural Networks

Neural networks define a type of machine learning algorithms that mimic the neurotic grids present on our brains. The most common form of this algorithms is Backpropagation. In a nutshell a neural network is comprised of a set of units (neurons) which are connected to each other with associated weights. During the training phase these weights are adjusted so as to predict the correct class or value for the input values. While very promising, this algorithm as the setback of producing models (the network) which are very hard if not impossible to understand.

CART

CART is part of a set of algorithms that make use of the decision tree models used in classification. In these cases, however, the leaf nodes consist of continuous values rather than simply the class one wants to predict.

Random Forest

Random Forest is a boosting method for the just mentioned algorithm. In this case, several regression trees are created for randomly selected sets of attributes from the original attribute space. While usually more effective than CART, this algorithm is obviously considerably more computationally heavy.

PPR

Projection Pursuit Regression [Friedman and Stuetzle, 1981], combines techniques from the above mentioned algorithms. Basically PPR tries to model a regression surface as a progressive sum of general functions of linear combinations of the available attributes (much like other boosting methods). Due to its computational weight, this algorithm has been unrightfully ignored until recently.

M5

Much like CART, M5 makes use of tree-based models, but is able to stipulate multivariate linear models both at its leaves and their parent nodes.

Parameter tuning was performed using carets *tuneLength* parameter of 4 and 8 as well as its default value. As seen in Figure 4.9, parameter tuning did not yield significantly higher results. However the training time of these 3 situations is considerably different (2.7, 3.5 and 6.1 hours respectively). To save time, without jeopardizing the potential benefits of parameter tuning, every experiment was run with a *tuneLength* value of 4.

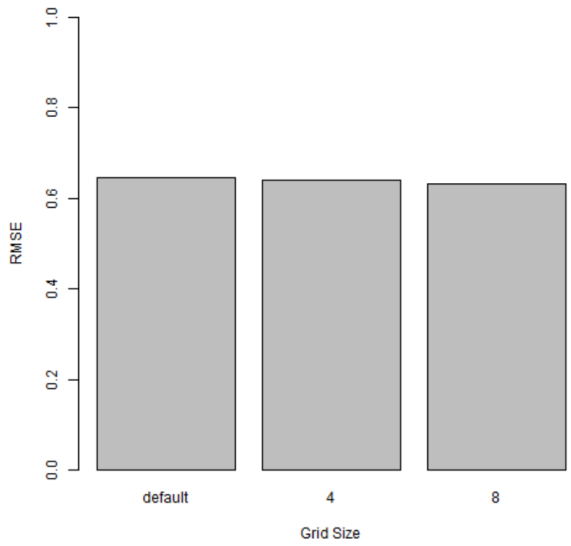

Figure 4.9: RMSE of Random Forest for CPL with different parameter values. Runtime for each situation was 2.7, 3.5 and 6.1 hours respectively.

The algorithms were run on R (v3.1.3), using the caret library and other smaller auxiliary libraries such as *hydroGOF*, *SDMTools* and *CORElearn*. The results for both experiments are presented and discussed in the next section.

4.3 Results and Discussion

As described above, in the first experiment, we tested several machine learning algorithms over a subset of the described metafeatures. The results of such tests can be found in greater detail on A.1, A.5, A.9, A.13. From these results we selected the algorithm that, for each component, performed better. The metric used for the selection was the average difference of the RMSE for our model versus that of the RMSE for the base model (both Mean and Median) over each category and considering the number of instances involved (vs Mean (per category) - Weighted and vs Median (per category) - Weighted respectively). In annex, it is possible to find how these models scored over the entire dataset without considering each category individually. In these cases, the base model is the mean or median of the acceptance rate of every category and iteration. Since this value somewhat misrepresents the actual performance of our models, we decided to focus on the remaining metrics. In Table 4.1, results from the first experiment are presented.

Component	**Best Algorithm**	**vs Mean (per category)**	**vs Median (per category)**	**vs Mean (per category) - Weighted**	**vs Median (per category) - Weighted**	**Median of vs Mean (per category)**	**Median of vs Median (per category)**
CPL	M5	16.433334 +/- 74.607998	18.800661 +/- 75.442796	16.575102 +/- 72.759929	18.858014 +/- 73.621033	0.795580	0.920185
SEAL	SVM	1.306163 +/- 1.616603	1.343011 +/- 1.797465	1.673811 +/- 2.065213	1.771162 +/- 2.336188	0.920173	0.834854
CMC	Random Forest	1.580437 +/- 1.762708	1.709427 +/- 2.507000	1.836546 +/- 2.070779	1.970727 +/- 2.740827	1.134150	1.047657
Ontology Extension	KNN	0.539202 +/- 0.755699	0.553160 +/- 0.806748	0.707829 +/- 0.663050	0.743547 +/- 0.800977	0.249309	0.249309

Table 4.1: Results of each component for the 1º experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

From Table 4.1 it is clear that there are some categories which are affecting our results. For CPL, for example, the average performance, considering the number of instances, is 16.575102 while most of its categories have a median performance of just 0.795580 (Median of the performance).

When plotting the performance of each component versus its acceptance rate (Figure 4.10) it becomes clear that, in most cases, the categories that are being scored least are those that have lower acceptance rates.

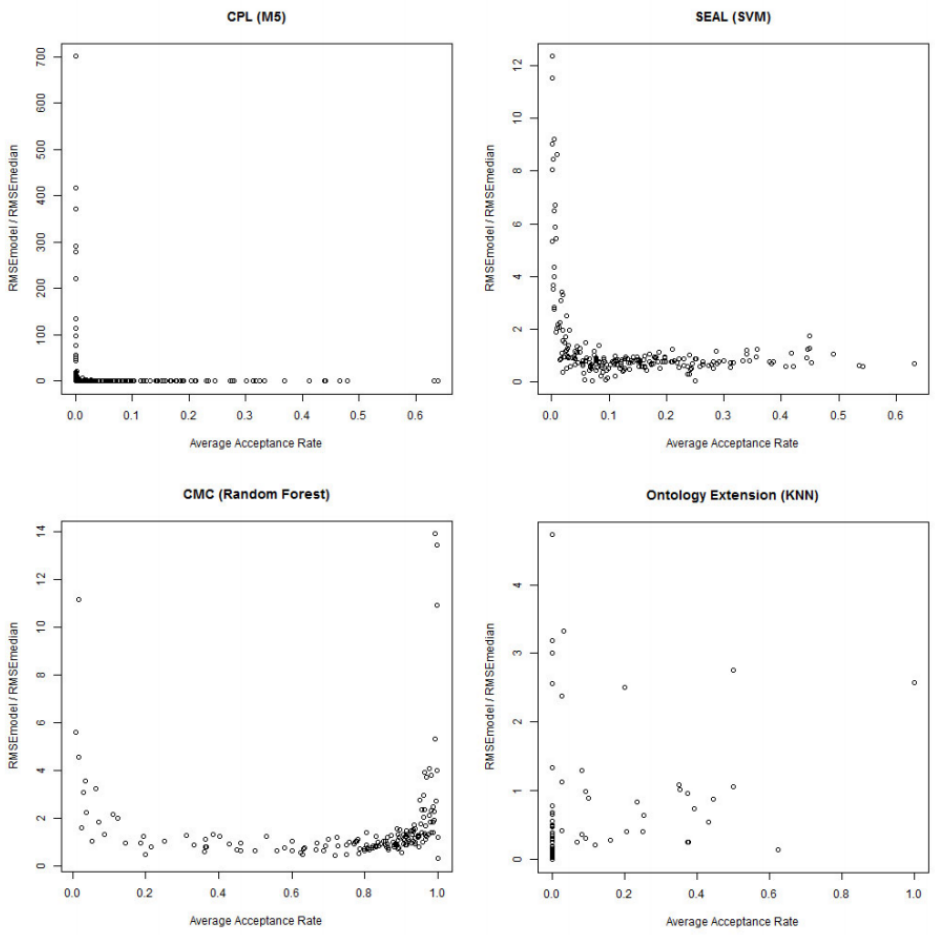

Figure 4.10: Average acceptance rate of the testing set compared to the performance of the model for the first experiment.

In light of this unexpected behaviour, it became important to find out what was causing such poor performance values. In Figure 4.11 we plotted the acceptance rate of the training set against that of the testing set. As can be seen, there is a large number of categories, in CPL and SEAL, that have an acceptance rate close to zero in both the training and testing set. Furthermore, it seems that having an acceptance rate of zero (or very close to zero) in the training set implies that you will find an acceptance rate of zero in the testing set. It seems that, for some reason, NELL is unable to find instances that are accepted for these categories no matter what.

One could attribute this lack of acceptance to a lower number of instances proposed. Indeed, if the algorithm proposes 500 instances in one iteration, we would expect at least one of these instances to be accepted. If, however, only one instance is proposed then it is reasonable that we will find an acceptance rate of zero in that iteration. In Figure 4.12 we plot the average number of instances proposed by each category versus its acceptance rate. If the just described behaviour was to be true, we would expect to see a lower acceptance rate when categories propose less instances. In fact, the opposite behaviour is observed.

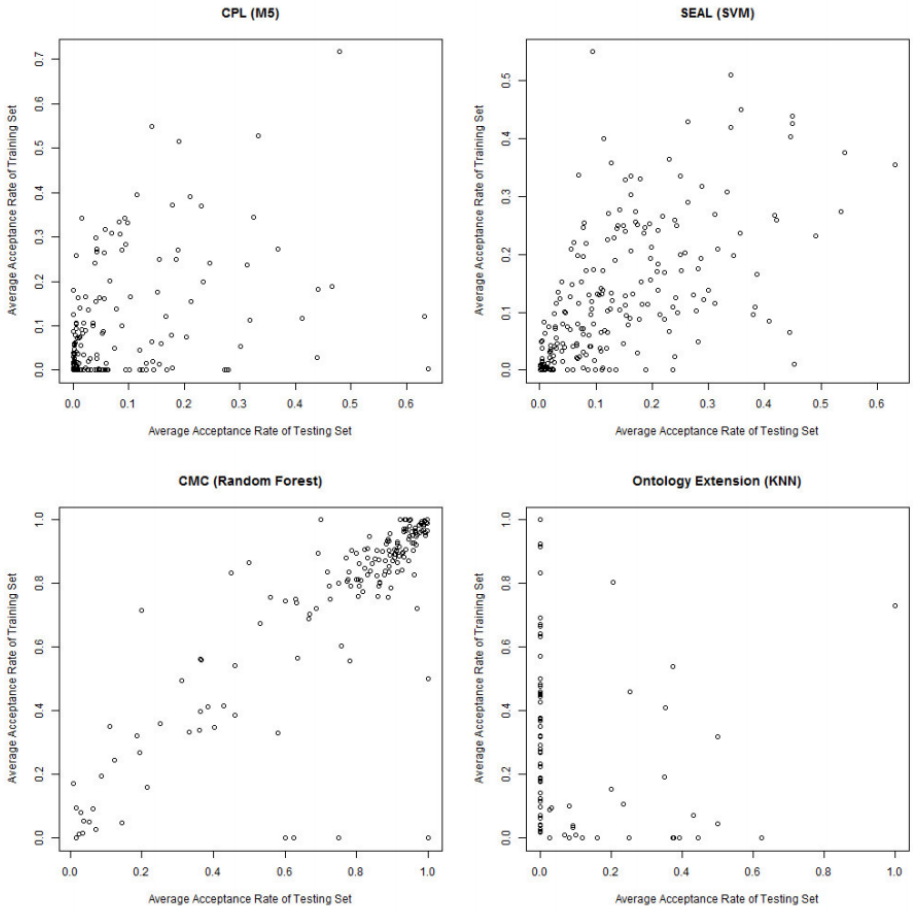

Figure 4.11: Average acceptance rate of the training set compared to average acceptance rate of the testing set.

It seems that, no matter what, some categories experience a very low acceptance rate. It might be that the initial seeds used for those categories are not good enough to coldstart NELLs learning procedure. On the other hand, it is also likely that the instances proposed for one category, by different components, have very different acceptance rates. Indeed, while CPL is very good at extracting instances for one category, SEAL might be unable to do so and vice-versa. In the future, it might be interesting to see how many of these categories overlap amongst different components.

Some other odd behaviours are also worth exploring. Not unexpectedly, CMC not only promotes a very high number of instances, but also does so in a predictable manner. In fact the higher the acceptance rate in the training set, the higher it will be in the testing set. Since our base model operates by that same principle it is no suprise that it performs very well in this situation. Our model, that takes no assumption over what category is being analysed or how it fared on it so far, is thus easily outperformed. Lastly, in Figure 4.11, Ontology Extension displays an abnormal behaviour. Probably by chance, instances proposed on the testing set have almost no chance of being promoted even though, in the past, iterations have had high/normal acceptance rates. A model

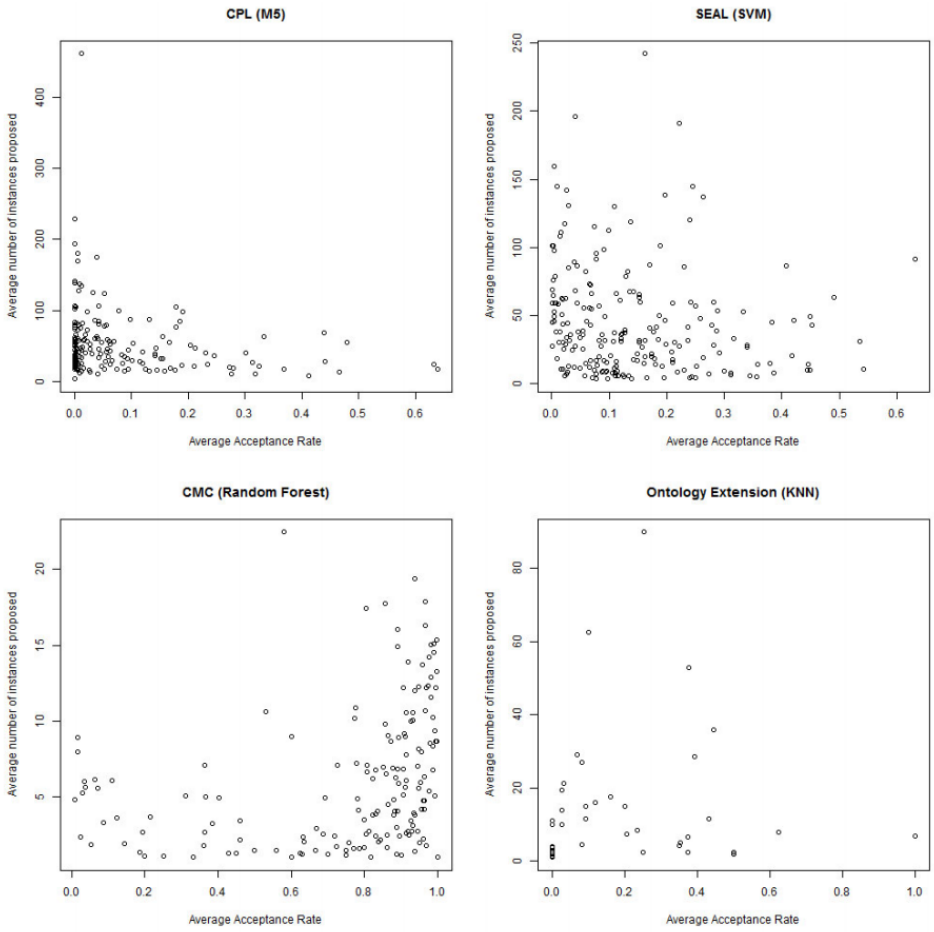

Figure 4.12: Average acceptance rate compared to number of instances proposed in the testing set.

prone to lower values of acceptance rate would easily outperform our base model and it is very likely that this is what is happening.

Armed with the knowledge that some categories are bound to have an acceptance rate of zero, we divised a model that would treat categories with very low acceptance rates differently than those with higher/normal acceptance rates. In a nutshell, the pseudo-code for such model would be:

Algorithm 1 Threshold Method

```
1: procedure PREDICTTESTACCEPTANCERATE
2:     if averageAcceptanceRateInTraining < threshold then
3:         return baseModelPrediction()
4:     else
5:         return ourModelPrediction()
```

From Figure 4.11 one can extrapolate the value of threshold (in our case we used an acceptance

rate of 0.05[4]) but it should be noted that this is not entirely acceptable. Since we are solely using the testing set for measuring performance, we should not use it to define a model beforehand. One possible way of finding the value of this threshold would be to divide the training set in much the same way as we divided the original set (70/30) and correlate those categories that show a very low acceptance rate in both sets. However, some precautions should be taken in order to properly identify those categories in which instances are never promoted. Firstly, if instances are proposed on a small number of iterations in the training set, it is unwise to think that, even if these instances are never promoted, this behaviour is representative of an inability to produce acceptable instances. In the very same way, even if there are a lot of iterations where instances are proposed, if, in these iterations, the median number of instances is very small, then it is likely that we observe a mean lower acceptance rate.

It is also worth noting that, by separately evaluating categories that are unable to propose promotable instances than those who are, we are implicity stating that these categories will never amount to anything in the future. It is very likely, however, that that does not stay true for very long since NELL is constantly learning and being improved.

The results for our alternative model are detailed on Table 4.2 and can be found in greater detail on A.2, A.6, A.10, A.14.

Component	**Best Algorithm**	**vs Mean (per category)**	**vs Median (per category)**	**vs Mean (per category) - Weighted**	**vs Median (per category) - Weighted**	**Median of vs Mean (per category)**	**Median of vs Median (per category)**
CPL	Random Forest	0.532123 +/- 0.360563	1.212594 +/- 1.897533	0.488096 +/- 0.343443	1.034096 +/- 1.505701	0.519757	0.640324
SEAL	Random Forest	0.794742 +/- 0.303365	0.821016 +/- 0.633219	0.782198 +/- 0.305875	0.841350 +/- 0.727263	0.799948	0.711370
CMC	Random Forest	1.488683 +/- 1.200311	1.627613 +/- 2.197947	1.688288 +/- 1.386464	1.828385 +/- 2.299364	1.136798	1.048700
Ontology Extension	M5	0.458968 +/- 0.790488	0.474340 +/- 0.794315	0.646042 +/- 0.889072	0.670701 +/- 0.899573	0.147826	0.164994

Table 4.2: Results of each component for the first experiment considering the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

As expected, this new method yields much better results. For both CPL and SEAL, there are still, however, some categories on which our model is under-performing. As can be seen in Figure 4.13, specially for CPL, these categories are still associated with lower acceptance rates.

From this first experiment, it is clear that, were we able to detect those categories for which no instance is ever promoted, our model would be able to perform much better. Coupled with the fact that CMC (the component with worst performance) is only capable to deal with category type

[4]To save time the value of threshold was manually extrapolated from the plots.

instances, we decided to focus our attention on metafeatures that described the past activity of the components rather than explore graph related features as we had done so far.

The results of the second experiment can are detailed on Table 4.3. Since we are already aware of the presence of outliers, the results presented in 4.3 are those of our threshold method. As in the first experiment the results of each algorithm over every component can be found in A.3, A.4, A.7, A.8, A.11, A.12, A.15, A.16.

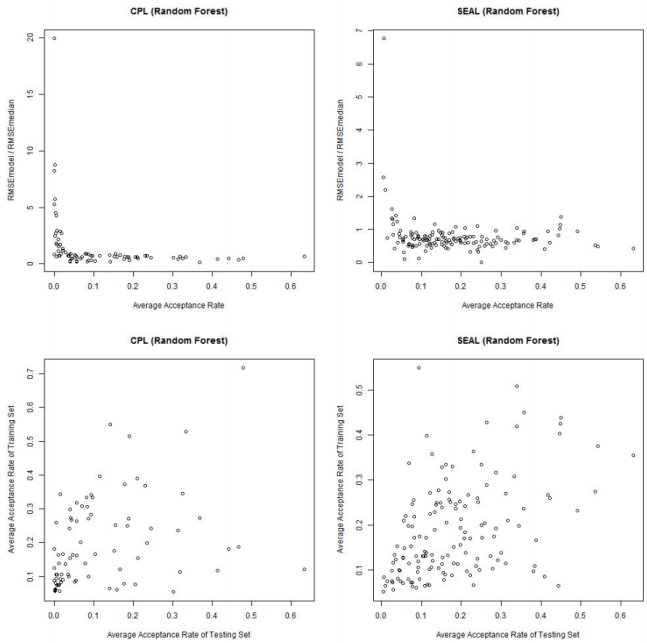

Figure 4.13: Acceptance Rate versus Performance (Top) and Acceptance Rate of Training set versus Acceptance Rate of Testing set (Bottom), for CPL and SEAL, after removing categories with an acceptance rate, in the training set, bellow 0.05.

As can be seen in Table 4.3, the performance of our models for each component did not improve significantly. In fact, many, if not most, models created in the second experiment yield worse results than those in the first. This is a good indicator that the activity of a component in the past does not significantly affect its present behaviour. One clear objective of these new metafeatures (such as the average number of instances proposed and accepted in the last 14 iterations) was to try and predict which categories did not produce promotable instances. It is likely that an interval of 14 iterations is not enough time to correctly predict if instances are being accepted or not. One could propose that this interval be expanded so as to correctly represent the activity of the system. However, the introduction of these metafeatures introduces a lag in the system (at this point we can only start counting from iteration 14 while before we could use every iteration from 0 to 250). It is also worth noting that the algorithms that suffered less from the addition of new metafeatures are those that are better suited to handle feature selection (such as regression tree and random forest).

Since the addition of activity related metafeatures brought little improvement to our system, it is likely that graph related features are the safest bet so far. As can be seen in 4.3 our models were

unable to produce better results than the base models for CMC even after adding new metafeatures. Since CMC shares not only a different role than the other 3 algorithms, but is also only able to deal with category type instances, it is probable that we won't be able to improve these results with our current strategy. Ontology Modifier on the other hand, seems to perform very well, but, as we seen before, it is likely being misrepresented because of our base model. On the other hand, both CPL and SEAL show promising results even though they are seemingly being held back by the categories that are unable to propose promotable instances.

Component	**Best Algorithm**	**vs Mean (per category)**	**vs Median (per category)**	**vs Mean (per category) - Weighted**	**vs Median (per category) - Weighted**	**Median of vs Mean (per category)**	**Median of vs Median (per category)**
CPL	Regression Tree	0.562039 +/- 0.284351	1.251212 +/- 2.123764	0.526642 +/- 0.287506	1.052983 +/- 1.517202	0.549472	0.744232
SEAL	Random Forest	**0.767042 +/- 0.270038**	**0.780708 +/- 0.595535**	**0.753606 +/- 0.275337**	**0.791049 +/- 0.692412**	**0.767994**	**0.698319**
CMC	M5	1.502553 +/- 1.062688	**1.614203 +/- 1.840125**	**1.666529 +/- 1.197274**	**1.784434 +/- 1.908445**	**1.125097**	1.06417
Ontology Extension	Random Forest	**0.431380 +/- 0.784002**	**0.436274 +/- 0.782606**	**0.631252 +/- 0.978876**	**0.630785 +/- 0.975331**	**0.130308**	**0.141460**

Table 4.3: Results of each component for the second experiment considering the threshold method. The RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category. The values in bold represent those which outperformed the first experiment.

At the moment, the safest route would be to first improve our threshold method to guarantee that we are currently identifying the just mentioned categories. As can be seen in Table 4.3, the threshold method brings little improvement to CMC and Ontology Modifier (probably because these components explicitly focus on a sub set of categories).

A second improvement to our system could be to tweak the predictions made by the models to correctly represent the number of instances accepted. A prediction of an acceptance rate of 42% for an iteration where only 3 instances have been proposed is not accurately representing our model. Rounding up the value of our prediction to whatever value makes sense (in this case the acceptance rate could only take the value of 0%, 33%, 66% or 100%) could improve our prediction model.

It is also important to note that the training of our model is done considering all categories without any previous knowledge. Since we have shown that some categories behave very differently to what is to be expected, the training of a new model could be done with a new training set that does not include these categories. However, since at the moment we are unable to precisely identify the misbehaving categories, this new training set could be lacking examples where acceptance rates are very low and thus not be able to generalize for such cases. Lastly, it is important to debate what new metafeatures should be added in the future. As discussed, graph-related features

seem to be our best bet. As mentioned in chapter 2.2, Prophet searches for open and close triangles to assess the accuracy of promoted instances. In very much the same way, we could take every proposed instance of every category for an iteration and create a graph on which we could search for cyclic paths. It is likely that categories involved in these cycles have instances with a higher chance of being promoted.

Chapter 5

Conclusions and Future Work

In this work we focused on filling a gap in NELL's monitoring ability. As it stands NELL is unable to evaluate the performance of its components in real time. On the other hand, the amount and diversity of the information involved calls for a new method of measuring such performance. Our goal for this project was to lean on the capabilities of metalearning to quickly analyze and predict the performance of such components. In doing so, we created prediction models for each components based on metafeatures of the data proposed by them. With these models, one can predict the quality, that is, the future acceptance rate of the information proposed by a component without waiting for confirmation from other components or even itself. The evaluation of these models was made by comparing the error of the prediction against the error of simpler models such as the average and median accuracy of the component.

However, as discussed in chapter 4, the models show very different results (the most promising being CPL and SEAL). For these components, our metalearning system was able to predict the future acceptance of the facts proposed better than both the median and average overall. For other components like CMC, this was not observed. For Ontology Extension, however, we observe a very high predictive capability, but, as we have shown, this is simply due to an erratic behavior of NELLs rather than a triumph of our model.

These preliminary results show that our metalearning approach is, at least for some components, a viable option to enhance NELL's learning capabilities. In the future, when more data is available, our solution could be applied to newer iterations, and encompass other components such as PRA and OpenEval. Furthermore, new metafeatures could be added to our model such has features regarding open triangles as used by prophet. Lastly new machine learning algorithms could be added and parameter variation could be further explored.

References

Harith Alani, Sanghee Kim, David E Millard, Mark J Weal, Paul H Lewis, Wendy Hall, and Nigel R Shadbolt. Automatic extraction of knowledge from web documents. 2003.

Kalousis Alexandros and Hilario Melanie. Model selection via meta-learning: a comparative study. *International Journal on Artificial Intelligence Tools*, 10(04):525–554, 2001.

Ana Paula Appel and Estevam Rafael Hruschka Jr. Prophet–a link-predictor to learn new rules on nell. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 917–924. IEEE, 2011.

Michele Banko and Oren Etzioni. Strategies for lifelong knowledge extraction from the web. In *Proceedings of the 4th international conference on Knowledge capture*, pages 95–102. ACM, 2007.

Hilan Bensusan and Christophe Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *Principles of Data Mining and Knowledge Discovery*, pages 325–330. Springer, 2000.

Ronald J Brachman and Tej Anand. The process of knowledge discovery in databases. In *Advances in knowledge discovery and data mining*, pages 37–57. American Association for Artificial Intelligence, 1996.

Pavel Brazdil, Carlos Soares, and Rui Pereira. Reducing rankings of classifiers by eliminating redundant classifiers. In *Progress in Artificial Intelligence*, pages 14–21. Springer, 2001.

Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

Pavel B Brazdil and Carlos Soares. A comparison of ranking methods for classification algorithm selection. In *Machine Learning: ECML 2000*, pages 63–75. Springer, 2000.

Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

Andrew Carlson, Justin Betteridge, Estevam R Hruschka Jr, and Tom M Mitchell. Coupling semi-supervised learning of categories and relations. In *Proceedings of the NAACL HLT 2009 Workshop on Semi-supervised Learning for Natural Language Processing*, pages 1–9. Association for Computational Linguistics, 2009.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010a.

REFERENCES

Andrew Carlson, Justin Betteridge, Richard C Wang, Estevam R Hruschka Jr, and Tom M Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 101–110. ACM, 2010b.

Philip K Chan and Salvatore J Stolfo. Experiments on multistrategy learning by meta-learning. In *Proceedings of the second international conference on information and knowledge management*, pages 314–323. ACM, 1993.

Jinxiu Chen, Donghong Ji, Chew Lim Tan, and Zhengyu Niu. Relation extraction using label propagation based semi-supervised learning. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 129–136. Association for Computational Linguistics, 2006.

Xinlei Chen, Ashish Shrivastava, and Arpan Gupta. Neil: Extracting visual knowledge from web data. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1409–1416. IEEE, 2013.

Bing Cheng and D Michael Titterington. Neural networks: A review from a statistical perspective. *Statistical science*, pages 2–30, 1994.

Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and information systems*, 1(1):5–32, 1999.

James R Curran, Tara Murphy, and Bernhard Scholz. Minimising semantic drift with mutual exclusion bootstrapping. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, volume 6, 2007.

Miguel Delgado, Daniel SáNchez, MarıA J MartıN-Bautista, and MarıA-Amparo Vila. Mining association rules with improved semantics in medical databases. *Artificial Intelligence in Medicine*, 21(1):241–245, 2001.

Vitor Hugo Gonçalves dos Santos. Never ending language metalearning: model management for cmus readtheweb project. 2014.

Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

Jerome H Friedman and Werner Stuetzle. Projection pursuit regression. *Journal of the American statistical Association*, 76(376):817–823, 1981.

Jerome H Friedman, Forest Baskett, and Leonard J Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on computers*, (10):1000–1006, 1975.

Aldo Gangemi. A comparison of knowledge extraction tools for the semantic web. In *The semantic web: Semantics and big data*, pages 351–366. Springer, 2013.

Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. 2013.

Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

REFERENCES

Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

Melanie Hilario and Alexandros Kalousis. *Quantifying the resilience of inductive classification algorithms*. Springer, 2000.

Melanie Hilario and Alexandros Kalousis. *Fusion of meta-knowledge and meta-data for case-based model selection*. Springer, 2001.

Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

Petr Kadlec and Bogdan Gabrys. Architecture for development of adaptive on-line prediction models. *Memetic Computing*, 1(4):241–269, 2009.

Sotiris Kotsiantis and Dimitris Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

Jayant Krishnamurthy and Tom M Mitchell. Which noun phrases denote which concepts? In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 570–580. Association for Computational Linguistics, 2011.

Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 529–539. Association for Computational Linguistics, 2011.

T Mitchell, W Cohen, E Hruschka, P Talukdar, J Betteridge, A Carlson, B Dalvi, M Gardner, B Kisiel, J Krishnamurthy, et al. Never-ending learning. 2015.

Thahir P Mohamed, Estevam R Hruschka Jr, and Tom M Mitchell. Discovering relations between noun categories. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1447–1455. Association for Computational Linguistics, 2011.

Un Yong Nahm and Raymond J Mooney. A mutually beneficial integration of data mining and information extraction. In *AAAI/IAAI*, pages 627–632, 2000.

M Pavan and R Todeschini. New indices for analysing partial ranking diagrams. *Analytica chimica acta*, 515(1):167–181, 2004.

Saulo DS Pedro and Estevam R Hruschka Jr. Collective intelligence as a source for machine learning self-supervision. In *Proceedings of the 4th International Workshop on Web Intelligence & Communities*, page 5. ACM, 2012.

Saulo DS Pedro, Ana Paula Appel, and Estevam R Hruschka Jr. Autonomously reviewing and validating the knowledge base of a never-ending learning system. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1195–1204. International World Wide Web Conferences Steering Committee, 2013.

J Ross Quinlan and R Mike Cameron-Jones. Foil: A midterm report. In *Machine Learning: ECML-93*, pages 1–20. Springer, 1993.

Larry Rendell, Raj Seshu, and David Tcheng. More robust concept learning using dynamically-variable bias. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 66–78. University of California Irvine, California, 1987.

REFERENCES

Ellen Riloff, Rosie Jones, et al. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

Lorenza Saitta and Filippo Neri. Learning in the "real world". *Machine Learning*, 30(2-3):133–163, 1998.

Mehdi Samadi, Manuela M Veloso, and Manuel Blum. Openeval: Web information query evaluation. In *AAAI*. Citeseer, 2013.

Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, 2013.

Sharon Gower Small and Larry Medsker. Review of information extraction technologies and applications. *Neural Computing and Applications*, 25(3-4):533–548, 2014.

Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2008.

Carlos Soares and Pavel B Brazdil. Selecting parameters of svm using meta-learning and kernel matrix-based meta-features. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 564–568. ACM, 2006.

Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

Julien Subercaze and Christophe Gravier. Fop: Never-ending learner for multimedia knowledge extraction. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on*, volume 2, pages 459–466. IEEE, 2014.

Joaquin Vanschoren and Hendrik Blockeel. Towards understanding learning behavior. In *Proceedings of the Annual Machine Learning Conference of Belgium and The Netherlands, Benelearn*, pages 89–96, 2006.

Thiago Lima Vieira and Helena de Medeiros Caseli. Nebel: Never-ending bilingual equivalent learner. In *Human-Inspired Computing and Its Applications*, pages 99–103. Springer, 2014.

Richard C Wang and William W Cohen. Language-independent set expansion of named entities using the web. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 342–350. IEEE, 2007.

Richard C Wang and William W Cohen. Character-level analysis of semi-structured documents for set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1503–1512. Association for Computational Linguistics, 2009.

Cort J Willmott. Some comments on the evaluation of model performance. *Bulletin of the American Meteorological Society*, 63(11):1309–1313, 1982.

Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79, 2005.

Appendix A

Appendix

A.0.1 CPL Results

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.683806	0.664605	19.098067 +/- 92.835872	21.551451 +/- 93.631266	23.177866 +/- 96.733941	25.656029 +/- 97.389017	0.760799	0.867310
SVM	0.730791	0.710271	21.770815 +/- 82.310720	26.579746 +/- 85.224179	30.226230 +/- 100.243359	35.562181 +/- 102.479008	0.853356	0.929231
KNN	0.721706	0.701442	20.293310 +/- 112.952476	23.323029 +/- 113.342647	23.159995 +/- 112.507288	26.747862 +/- 112.991246	0.824133	0.965615
KNN (with RReliefF	0.716804	0.696677	17.938389 +/- 84.020695	21.349295 +/- 84.929050	21.727653 +/- 85.469747	25.691388 +/- 86.433949	0.810087	0.980806
Neural Network	0.664796	0.646130	20.479423 +/- 94.987456	24.458286 +/- 97.505142	22.735667 +/- 98.663074	27.378885 +/- 101.056741	0.692466	0.892121
Regression Tree	0.680993	0.661872	15.972653 +/- 78.008957	18.296920 +/- 78.995959	16.222366 +/- 74.745631	18.687126 +/- 75.754683	0.771367	0.952426
Random Forest	0.647588	0.629405	18.660864 +/- 106.291270	21.710328 +/- 107.457732	18.801573 +/- 101.915976	22.009916 +/- 103.165473	0.674607	0.788877
PPR	0.647192	0.629020	21.950439 +/- 93.782211	25.885388 +/- 95.544811	27.054030 +/- 97.606712	31.583196 +/- 99.196202	0.690031	0.840649
M5	0.666435	0.647723	16.268908 +/- 84.854721	19.248498 +/- 87.805454	15.716287 +/- 79.193077	18.902145 +/- 82.458995	0.692838	0.787988

Table A.1: Results of CPL for the first experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.610747 +/- 0.328222	1.282219 +/- 1.572081	0.575976 +/- 0.348614	1.150786 +/- 1.373477	0.658627	0.731452
SVM	0.588667 +/- 0.304437	2.026184 +/- 6.242392	0.568068 +/- 0.273998	1.643276 +/- 4.504275	0.574984	0.747840
KNN	0.581098 +/- 0.266394	1.579171 +/- 3.893992	0.565413 +/- 0.258612	1.357693 +/- 2.823901	0.599007	0.823701
KNN (with RReliefF	0.567447 +/- 0.271224	1.673548 +/- 4.760991	0.568234 +/- 0.255115	1.447736 +/- 3.439072	0.583935	0.728925
Neural Network	0.570194 +/- 0.306398	1.409470 +/- 2.424185	0.597685 +/- 0.298433	1.444695 +/- 1.886254	0.588514	0.736389
Regression Tree	0.569739 +/- 0.304630	1.265893 +/- 2.158623	0.526238 +/- 0.305337	1.063672 +/- 1.566037	0.587873	0.778973
Random Forest	0.532123 +/- 0.360563	1.212594 +/- 1.897533	0.488096 +/- 0.343443	1.034096 +/- 1.505701	0.519757	0.640324
PPR	0.541053 +/- 0.315419	1.491029 +/- 3.409273	0.511726 +/- 0.299139	1.248169 +/- 2.535795	0.538911	0.667265
M5	0.537119 +/- 0.345906	1.042168 +/- 1.356401	0.480030 +/- 0.331875	0.897080 +/- 1.113755	0.526055	0.695697

Table A.2: Results of CPL for the first experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.688881	0.669539	21.872218 +/- 93.795982	25.356653 +/- 94.642609	30.186382 +/- 108.456051	34.390898 +/- 108.892564	0.770703	0.934543
SVM	0.821044	0.797990	30.187757 +/- 111.758522	38.619579 +/- 118.708786	46.124048 +/- 154.509652	56.182557 +/- 159.699801	0.955860	1.080963
KNN	0.896929	0.871745	21.156550 +/- 96.065156	26.249569 +/- 99.355288	24.683786 +/- 98.847176	29.885238 +/- 101.852890	1.016366	1.088759
KNN (with RReliefF	0.918274	0.892490	22.354462 +/- 91.577703	29.546606 +/- 98.513317	27.948602 +/- 99.998122	35.675327 +/- 106.317480	1.045122	1.141844
Neural Network	0.913144	0.887504	46.748131 +/- 193.044793	55.757061 +/- 198.299646	75.420567 +/- 264.879739	87.559636 +/- 268.464842	0.947133	1.284484
Regression Tree	0.672856	0.653963	11.911602 +/- 50.470148	13.612241 +/- 50.722278	13.081225 +/- 50.719268	14.927326 +/- 50.977056	0.731439	0.950845
Random Forest	0.628792	0.611137	16.599529 +/- 69.093739	20.027084 +/- 71.078163	18.147729 +/- 68.049589	21.725191 +/- 70.030653	0.680378	0.833679
PPR	0.883760	0.858945	77.409498 +/- 527.849902	52.833410 +/-230.6054673	260.32514 +/-1137.402356	91.225234 +/-320.8422501	0.831466	1.049204
M5	0.766538	0.745014	16.433334 +/- 74.607998	18.800661 +/- 75.442796	16.575102 +/- 72.759929	18.858014 +/- 73.621033	0.795580	0.920185

Table A.3: Results of CPL for the second experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.608399 +/- 0.332449	1.540062 +/- 2.615879	0.682194 +/- 0.359808	1.813790 +/- 2.397777	0.631076	0.709275
SVM	0.727244 +/- 0.327376	2.868345 +/- 9.027183	0.763815 +/- 0.297610	2.452675 +/- 6.535891	0.752186	0.868191
KNN	0.781270 +/- 0.382065	2.484707 +/- 7.265534	0.749490 +/- 0.417237	1.935097 +/- 5.230470	0.808294	0.950087
KNN (with RReliefF	0.808665 +/- 0.386772	2.604261 +/- 7.309138	0.793294 +/- 0.422835	2.090570 +/- 5.277521	0.840490	0.955217
Neural Network	0.893222 +/- 0.573591	2.494606 +/- 4.601932	1.173868 +/- 0.695313	3.459856 +/- 4.715150	0.758745	0.929716
Regression Tree	0.562039 +/- 0.284351	1.251212 +/- 2.123764	0.526642 +/- 0.287506	1.052983 +/- 1.517202	0.549472	0.744232
Random Forest	0.572885 +/- 0.336577	1.493219 +/- 2.651206	0.526668 +/- 0.324821	1.210607 +/- 1.983691	0.577265	0.715668
PPR	0.782968 +/- 0.561528	1.827168 +/- 2.808864	1.007404 +/- 0.696546	2.656936 +/- 3.232294	0.715103	0.830698
M5	0.636202 +/- 0.418744	1.359302 +/- 2.284778	0.607697 +/- 0.402001	1.230542 +/- 1.803324	0.605636	0.844642

Table A.4: Results of CPL for the second experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

A.0.2 SEAL Results

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.804646	0.704666	1.664334 +/- 2.885652	1.766828 +/- 3.281773	2.302684 +/- 3.725690	2.500640 +/- 4.312587	0.857355	0.770650
SVM	0.860423	0.753511	1.306163 +/- 1.616603	1.343011 +/- 1.797465	1.673811 +/- 2.065213	1.771162 +/- 2.336188	0.920173	0.834854
KNN	0.865752	0.758178	1.789797 +/- 3.267520	1.894087 +/- 3.659459	2.459569 +/- 4.134883	2.660033 +/- 4.696066	0.932852	0.859812
KNN (with RReliefF	0.891593	0.780809	1.707017 +/- 2.806757	1.791153 +/- 3.126935	2.289097 +/- 3.527440	2.458239 +/- 3.976590	0.959216	0.883304
Neural Network	0.813752	0.712640	1.550744 +/- 2.621395	1.640667 +/- 2.979172	2.096385 +/- 3.351608	2.269772 +/- 3.860764	0.875935	0.786921
Regression Tree	0.800391	0.700939	1.775189 +/- 3.390425	1.900975 +/- 3.821829	2.536536 +/- 4.444301	2.777979 +/- 5.082831	0.835338	0.767866
Random Forest	0.776242	0.679791	1.646934 +/- 2.989794	1.750963 +/- 3.345827	2.381472 +/- 4.066114	2.588248 +/- 4.582910	0.825273	0.771523
PPR	0.792196	0.693762	1.560192 +/- 2.612970	1.655539 +/- 2.994248	2.159865 +/- 3.384035	2.345232 +/- 3.941296	0.847756	0.766908
M5	0.781638	0.684516	1.537174 +/- 2.507690	1.639016 +/- 2.857210	2.129907 +/- 3.268572	2.317862 +/- 3.738463	0.837166	0.770608

Table A.5: Results of SEAL for the first experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.825959 +/- 0.268811	0.849988 +/- 0.643357	0.837961 +/- 0.273593	0.893378 +/- 0.751848	0.830253	0.740729
SVM	0.872589 +/- 0.291658	0.861638 +/- 0.447528	0.880096 +/- 0.276274	0.896696 +/- 0.505567	0.884778	0.816166
KNN	0.902988 +/- 0.285749	0.925814 +/- 0.620591	0.912915 +/- 0.269577	0.960620 +/- 0.679468	0.880415	0.800080
KNN (with RReliefF	0.937409 +/- 0.274174	0.951698 +/- 0.567563	0.978249 +/- 0.261384	1.018067 +/- 0.625283	0.919748	0.822776
Neural Network	0.833037 +/- 0.258968	0.847323 +/- 0.580392	0.859205 +/- 0.246548	0.902247 +/- 0.670963	0.844695	0.751903
Regression Tree	0.822936 +/- 0.298051	0.859085 +/- 0.730392	0.814451 +/- 0.326726	0.890978 +/- 0.868165	0.809419	0.732744
Random Forest	0.794742 +/- 0.303365	0.821016 +/- 0.633219	0.782198 +/- 0.305875	0.841350 +/- 0.727263	0.799948	0.711370
PPR	0.813533 +/- 0.268022	0.832240 +/- 0.587642	0.826362 +/- 0.264142	0.874713 +/- 0.677937	0.831897	0.735103
M5	0.806064 +/- 0.289789	0.828461 +/- 0.611707	0.802161 +/- 0.297386	0.850829 +/- 0.694476	0.807770	0.716255

Table A.6: Results of SEAL for the first experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.776164	0.679723	1.509475 +/- 2.510351	1.588097 +/- 2.779124	1.961215 +/- 3.118856	2.106649 +/- 3.505811	0.838377	0.773765
SVM	0.812214	0.711293	1.306740 +/- 1.675719	1.348570 +/- 1.843871	1.663084 +/- 2.048172	1.759194 +/- 2.283765	0.887827	0.784884
KNN	0.938898	0.822236	1.926246 +/- 3.499147	2.032347 +/- 3.806277	2.496970 +/- 4.360756	2.691258 +/- 4.802502	1.014116	0.951548
KNN (with RReliefF	0.977443	0.855992	1.931275 +/- 3.298295	2.035398 +/- 3.665802	2.538306 +/- 4.147199	2.726893 +/- 4.640032	1.031308	0.961449
Neural Network	0.772111	0.676173	1.202992 +/- 1.794714	1.247970 +/- 1.960417	1.488979 +/- 2.226205	1.580758 +/- 2.453997	0.846824	0.781395
Regression Tree	0.781848	0.684700	1.567266 +/- 2.894034	1.660814 +/- 3.179521	2.208254 +/- 3.908638	2.387478 +/- 4.306108	0.833859	0.774812
Random Forest	0.747642	0.654745	1.333270 +/- 2.030131	1.406305 +/- 2.309208	1.782574 +/- 2.852482	1.920985 +/- 3.185813	0.816263	0.730439
PPR	0.759130	0.664805	1.291080 +/- 2.046801	1.342932 +/- 2.198493	1.638641 +/- 2.538386	1.739996 +/- 2.743332	0.856944	0.777767
M5	0.766138	0.670942	1.329088 +/- 2.084378	1.388533 +/- 2.318167	1.725237 +/- 2.696427	1.837569 +/- 3.020992	0.843518	0.757865

Table A.7: Results of SEAL for the second experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.818445 +/- 0.251809	0.835061 +/- 0.569757	0.833591 +/- 0.245893	0.871061 +/- 0.648039	0.817870	0.731719
SVM	0.833472 +/- 0.274408	0.832190 +/- 0.501215	0.869513 +/- 0.251006	0.893765 +/- 0.567831	0.855696	0.763093
KNN	1.008477 +/- 0.293466	1.037676 +/- 0.730367	1.003804 +/- 0.287956	1.058794 +/- 0.813460	0.985812	0.874813
KNN (with RReliefF	1.040361 +/- 0.347662	1.078098 +/- 0.891336	1.050536 +/- 0.352175	1.119648 +/- 0.999679	1.007547	0.894371
Neural Network	0.817329 +/- 0.242595	0.814509 +/- 0.428914	0.849338 +/- 0.232640	0.862008 +/- 0.474982	0.816019	0.754292
Regression Tree	0.813767 +/- 0.273152	0.839891 +/- 0.676910	0.816946 +/- 0.276989	0.872742 +/- 0.794858	0.799303	0.724817
Random Forest	0.767042 +/- 0.270038	0.780708 +/- 0.595535	0.753606 +/- 0.275337	0.791049 +/- 0.692412	0.767994	0.698319
PPR	0.807276 +/- 0.249344	0.810476 +/- 0.467903	0.832855 +/- 0.235755	0.851762 +/- 0.519961	0.814835	0.735956
M5	0.809401 +/- 0.268337	0.813759 +/- 0.473544	0.850854 +/- 0.256572	0.868592 +/- 0.509889	0.818840	0.729808

Table A.8: Results of SEAL for the second experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

A.0.3 CMC Results

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.845952	0.694676	2.088160 +/- 1.959981	2.203798 +/- 2.550856	2.596557 +/- 2.287632	2.719668 +/- 2.807433	1.312334	1.235787
SVM	0.786329	0.645714	1.737932 +/- 1.622098	1.855749 +/- 2.295243	2.027253 +/- 1.871105	2.145614 +/- 2.431643	1.156426	1.073836
KNN	0.783583	0.643459	1.778940 +/- 1.781140	1.933011 +/- 2.639518	2.069054 +/- 2.077423	2.213862 +/- 2.790101	1.211191	1.132798
KNN (with RReliefF	0.783583	0.643459	1.778940 +/- 1.781140	1.933011 +/- 2.639518	2.069054 +/- 2.077423	2.213862 +/- 2.790101	1.211191	1.132798
Neural Network	0.766146	0.629141	1.813990 +/- 1.777696	1.942521 +/- 2.536490	2.205037 +/- 2.244944	2.351610 +/- 2.937951	1.220562	1.106755
Regression Tree	0.762209	0.625908	1.786718 +/- 1.816397	1.922158 +/- 2.670225	2.131590 +/- 2.254900	2.290640 +/- 3.074311	1.216910	1.122272
Random Forest	0.699923	0.574760	1.580437 +/- 1.762708	1.709427 +/- 2.507000	1.836546 +/- 2.070779	1.970727 +/- 2.740827	1.134150	1.047657
PPR	0.772043	0.633983	1.833583 +/- 1.754469	1.966264 +/- 2.555590	2.235445 +/- 2.141028	2.380054 +/- 2.862201	1.244349	1.133484
M5	0.748519	0.614666	1.641333 +/- 1.623908	1.743604 +/- 2.352998	1.879075 +/- 1.930362	1.984904 +/- 2.542184	1.190303	1.108684

Table A.9: Results of CMC for the first experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	2.021556 +/- 1.776914	2.148690 +/- 2.454924	2.491565 +/- 2.098261	2.622367 +/- 2.680629	1.305163	1.23418
SVM	1.660836 +/- 1.343596	1.788394 +/- 2.150013	1.923870 +/- 1.566213	2.047807 +/- 2.228517	1.154969	1.072644
KNN	1.703374 +/- 1.366121	1.868776 +/- 2.430514	1.942817 +/- 1.572766	2.093735 +/- 2.467013	1.218378	1.133078
KNN (with RReliefF	1.703374 +/- 1.366121	1.868776 +/- 2.430514	1.942817 +/- 1.572766	2.093735 +/- 2.467013	1.218378	1.133078
Neural Network	1.764587 +/- 1.694146	1.904347 +/- 2.522379	2.130155 +/- 2.165967	2.284181 +/- 2.902850	1.23053	1.108316
Regression Tree	1.722489 +/- 1.562321	1.868877 +/- 2.553535	2.019518 +/- 1.973668	2.185908 +/- 2.904827	1.226561	1.127441
Random Forest	1.488683 +/- 1.200311	1.627613 +/- 2.197947	1.688288 +/- 1.386464	1.828385 +/- 2.299364	1.136798	1.048700
PPR	1.786655 +/- 1.650709	1.930526 +/- 2.530412	2.157882 +/- 2.030461	2.309657 +/- 2.805805	1.248468	1.141323
M5	1.703374 +/- 1.366121	1.868776 +/- 2.430514	1.942817 +/- 1.572766	2.093735 +/- 2.467013	1.218378	1.133078

Table A.10: esults of CMC for the first experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.846264	0.694932	2.011426 +/- 2.320525	2.121645 +/- 2.773072	2.516342 +/- 2.983227	2.632474 +/- 3.329710	1.303702	1.240782
SVM	0.772303	0.634197	1.640753 +/- 1.385423	1.700736 +/- 1.785316	1.929002 +/- 1.644618	1.991066 +/- 1.979034	1.158168	1.090635
KNN	0.847634	0.696057	1.913981 +/- 1.882683	2.006892 +/- 2.303999	2.293973 +/- 2.332756	2.372371 +/- 2.607382	1.277449	1.181635
KNN (with RReliefF	0.836528	0.686937	1.844448 +/- 2.020699	1.920335 +/- 2.333280	2.221105 +/- 2.543098	2.280782 +/- 2.729504	1.202265	1.152038
Neural Network	0.772890	0.634679	1.650240 +/- 1.605260	1.735800 +/- 2.157885	1.922895 +/- 2.057439	2.014358 +/- 2.532417	1.183734	1.140073
Regression Tree	0.787825	0.646943	1.862874 +/- 2.018278	2.031241 +/- 3.098861	2.244869 +/- 2.531430	2.446458 +/- 3.612960	1.240350	1.156291
Random Forest	0.683616	0.561369	1.532442 +/- 1.429415	1.620396 +/- 1.936590	1.798372 +/- 1.837010	1.902582 +/- 2.306175	1.141618	1.087466
PPR	0.807393	0.663012	1.936868 +/- 2.827696	2.059549 +/- 3.303607	2.492842 +/- 3.899372	2.576430 +/- 4.228596	1.226111	1.147364
M5	0.738413	0.606367	1.569872 +/- 1.351470	1.672187 +/- 1.983496	1.774378 +/- 1.544740	1.886153 +/- 2.120261	1.132897	1.064814

Table A.11: Results of CMC for the second experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	1.934113 +/- 2.165288	2.055602 +/- 2.679582	2.402200 +/- 2.837145	2.525864 +/- 3.218619	1.302358	1.240017
SVM	1.564616 +/- 1.178238	1.632069 +/- 1.660088	1.831358 +/- 1.435263	1.898378 +/- 1.825613	1.157755	1.089438
KNN	1.822217 +/- 1.580167	1.924965 +/- 2.098636	2.161096 +/- 2.022237	2.245433 +/- 2.350994	1.276689	1.177898
KNN (with RReliefF	1.844448 +/- 2.020699	1.920335 +/- 2.333280	2.221105 +/- 2.543098	2.280782 +/- 2.729504	1.202265	1.152038
Neural Network	1.587585 +/- 1.459193	1.681750 +/- 2.086960	1.831625 +/- 1.913672	1.928729 +/- 2.436631	1.163447	1.137252
Regression Tree	1.803680 +/- 1.779969	1.984508 +/- 3.007012	2.136967 +/- 2.265855	2.346674 +/- 3.467987	1.236842	1.145194
Random Forest	1.483433 +/- 1.223373	1.579234 +/- 1.821649	1.708570 +/- 1.622611	1.818172 +/- 2.158782	1.148346	1.109064
PPR	1.936868 +/- 2.827696	2.059549 +/- 3.303607	2.441708 +/- 3.870458	2.576430 +/- 4.228596	1.226111	1.147364
M5	1.502553 +/- 1.062688	1.614203 +/- 1.840125	1.666529 +/- 1.197274	1.784434 +/- 1.908445	1.125097	1.06417

Table A.12: Results of CPL for the second experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

A.0.4 Ontology Extension Results

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.772141	0.923698	0.641444 +/- 0.950924	0.651185 +/- 0.952206	0.902576 +/- 1.012952	0.885840 +/- 1.002788	0.162332	0.183006
SVM	0.721773	0.863444	0.626894 +/- 0.918219	0.634201 +/- 0.916636	0.886217 +/- 0.983625	0.884351 +/- 0.980916	0.302600	0.307995
KNN	0.745172	0.891436	0.539202 +/- 0.755699	0.553160 +/- 0.806748	0.707829 +/- 0.663050	0.743547 +/- 0.800977	0.249309	0.249309
KNN (with RReliefF	0.751992	0.899595	0.530710 +/- 0.787880	0.549226 +/- 0.812188	0.824555 +/- 0.760371	0.846824 +/- 0.808346	0.113960	0.114566
Neural Network	0.713193	0.853181	0.634850 +/- 0.823548	0.651544 +/- 0.830100	0.797675 +/- 0.813961	0.799740 +/- 0.807113	0.273622	0.273622
Regression Tree	0.745473	0.891796	0.652509 +/- 0.962437	0.657308 +/- 0.948182	0.914502 +/- 1.072731	0.904158 +/- 1.052573	0.192921	0.206701
Random Forest	0.749864	0.897049	0.672011 +/- 0.907473	0.682302 +/- 0.907697	0.896637 +/- 0.984871	0.891901 +/- 0.979796	0.265699	0.255250
PPR	0.780129	0.933255	0.670244 +/- 0.950246	0.675681 +/- 0.945377	0.938227 +/- 1.026832	0.926085 +/- 1.012416	0.176197	0.179119
M5	0.766902	0.917432	0.628046 +/- 1.031028	0.632712 +/- 1.026955	0.893270 +/- 1.186123	0.887639 +/- 1.181448	0.186732	0.196749

Table A.13: Results of Ontology Extension for the first experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.491781 +/- 0.811892	0.512630 +/- 0.822665	0.698031 +/- 0.861401	0.700711 +/- 0.860439	0.122921	0.129478
SVM	0.495280 +/- 0.738849	0.513739 +/- 0.746661	0.774266 +/- 0.934770	0.803254 +/- 0.943761	0.228441	0.228441
KNN	0.483828 +/- 0.731018	0.507894 +/- 0.804989	0.786255 +/- 0.814406	0.875871 +/- 1.034711	0.184312	0.170517
KNN (with RReliefF	0.435076 +/- 0.758347	0.466859 +/- 0.798519	0.739414 +/- 0.829761	0.814398 +/- 0.931980	0.079848	0.079848
Neural Network	0.468135 +/- 0.686720	0.497502 +/- 0.709898	0.657344 +/- 0.763673	0.687069 +/- 0.775730	0.182480	0.182480
Regression Tree	0.456300 +/- 0.773261	0.474048 +/- 0.776757	0.702732 +/- 0.914798	0.717590 +/- 0.909206	0.126670	0.126670
Random Forest	0.503584 +/- 0.746990	0.523391 +/- 0.753114	0.724206 +/- 0.817795	0.741714 +/- 0.817072	0.193806	0.200753
PPR	0.527333 +/- 0.876175	0.542616 +/- 0.876552	0.772208 +/- 0.959413	0.785416 +/- 0.953823	0.126864	0.126864
M5	0.458968 +/- 0.790488	0.474340 +/- 0.794315	0.646042 +/- 0.889072	0.670701 +/- 0.899573	0.147826	0.164994

Table A.14: Results of Ontology Extension for the first experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.

	vs Mean	vs Median	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.747430	0.894138	0.543633 +/- 0.792714	0.545777 +/- 0.793188	0.909316 +/- 0.821283	0.905026 +/- 0.817245	0.144848	0.145487
SVM	0.714337	0.854548	0.638084 +/- 0.903162	0.649516 +/- 0.905229	0.895309 +/- 0.978370	0.900393 +/- 0.981111	0.311786	0.311786
KNN	0.754710	0.902846	0.631329 +/- 0.902559	0.642719 +/- 0.936292	0.712855 +/- 0.654043	0.745839 +/- 0.781176	0.295069	0.288328
KNN (with RReliefF	0.763577	0.913454	0.540134 +/- 0.831906	0.560953 +/- 0.857769	0.833564 +/- 0.772772	0.858342 +/- 0.826414	0.134680	0.134993
Neural Network	0.695263	0.831730	0.601097 +/- 0.771119	0.612095 +/- 0.773060	0.665762 +/- 0.749202	0.666289 +/- 0.749260	0.282819	0.270856
Regression Tree	0.750049	0.897271	0.636472 +/- 1.042983	0.644296 +/- 1.040609	0.828778 +/- 1.246858	0.824183 +/- 1.243400	0.221843	0.241151
Random Forest	0.715873	0.856386	0.618929 +/- 1.145472	0.616816 +/- 1.142209	0.813213 +/- 1.340342	0.798855 +/- 1.337508	0.181865	0.183362
PPR	0.704605	0.842906	0.489239 +/- 0.703169	0.495922 +/- 0.701300	0.646882 +/- 0.702283	0.640000 +/- 0.693947	0.174709	0.173762
M5	0.766138	0.936001	0.661873 +/- 1.175661	0.669639 +/- 1.174762	0.895167 +/- 1.242202	0.896623 +/- 1.242830	0.213050	0.242216

Table A.15: Results of Ontology Extension for the second experiment. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category and globally, considering the average acceptance rate of all categories as one (vs Mean and vs Median).

	vs Mean (per category)	vs Median (per category)	vs Mean (per category) - Weighted	vs Median (per category) - Weighted	Median of vs Mean (per category)	Median of vs Median (per category)
Linear Regression	0.428472 +/- 0.753516	0.436222 +/- 0.757118	0.881066 +/- 0.908343	0.897135 +/- 0.907979	0.085660	0.090870
SVM	0.502522 +/- 0.739204	0.526400 +/- 0.753904	0.800924 +/- 0.937075	0.843040 +/- 0.957367	0.229896	0.260414
KNN	0.482716 +/- 0.690981	0.505320 +/- 0.762604	0.762487 +/- 0.750251	0.847734 +/- 0.968884	0.192161	0.186018
KNN (with RReliefF	0.422791 +/- 0.708431	0.457817 +/- 0.757024	0.720658 +/- 0.765962	0.800488 +/- 0.890580	0.090909	0.090909
Neural Network	0.468579 +/- 0.714827	0.487683 +/- 0.720940	0.639778 +/- 0.900368	0.655363 +/- 0.898114	0.181721	0.184753
Regression Tree	0.439840 +/- 0.736828	0.457600 +/- 0.740922	0.668767 +/- 0.908417	0.683635 +/- 0.903424	0.126670	0.126670
Random Forest	0.431380 +/- 0.784002	0.436274 +/- 0.782606	0.631252 +/- 0.978876	0.630785 +/- 0.975331	0.130308	0.141460
PPR	0.422358 +/- 0.758629	0.438473 +/- 0.760577	0.701380 +/- 0.916788	0.714555 +/- 0.908107	0.093994	0.093994
M5	0.485722 +/- 0.839075	0.503393 +/- 0.842644	0.665317 +/- 0.786419	0.696793 +/- 0.797417	0.088793	0.093564

Table A.16: Results of Ontology Extension for the second experiment with the threshold method. RMSE of each model is compared against RMSE of base methods (Mean and Median) for each category.